NGUYỄN HỮU TRUNG

♦ ♦ ♦

TÀI LIỆU LUYỆN THI HỌC SINH GIỎI
MÔN: TIN 9 – THCS
(Tập 1)

Đà Nẵng, tháng 12 năm 2025

LỜI NÓI ĐẦU

Xin chào tất cả mọi người!

Theo bản thân tôi được biết thì việc bồi dưỡng học sinh giỏi của giáo viên và luyện thi để tham gia dự thi học sinh giỏi của học sinh là một trong những công việc mà hằng năm được nhiều giáo viên và học sinh quan tâm. Xuất phát từ thực tiễn đó, bằng tất cả tâm huyết của mình, bản thân tôi đã cố gắng tìm tòi, học hỏi và nghiên cứu để biên soạn ra bộ tài liệu này. Với mong muốn bộ tài liệu sẽ góp được một phần nhỏ vào việc tiết kiệm một chút ít thời gian biên soạn nội dung giảng dạy của quý thầy cô và đồng thời cũng cung cấp cho các em học sinh thêm một tài liệu tham khảo trong quá trình ôn luyện thi học sinh giỏi của mình.

Nội dung chính của tài liệu gồm hơn **700** trang giấy A4 (hơn bảy trăm trang giấy A4) được chia thành 2 phần:

+ **Phần 1** (Từ trang **06** đến trang **356**). Soạn ra lộ trình luyện thi học sinh giỏi Môn Tin Lớp 9 – THCS.

+ **Phần 2** (Từ trang **357** đến trang **733**). Luyện giải đề thi theo cấu trúc tỉnh.

- Cụ thể từng phần như sau:

+ **Phần 1. Lộ trình luyện thi:**

▪ Bao gồm **11** chủ đề với **33** nội dung bài học, tập trung vào các kiến thức trọng tâm có trong đề thi học sinh giỏi, đề thi vào các trường THPT chuyên và đề thi tin học trẻ bảng B (THCS).

▪ Nội dung của mỗi bài học được thiết kế theo các hoạt động tương tự như một tiết dạy học trên lớp theo các bước: Khởi động – Hình thành kiến thức mới – Luyện tập, vận dụng – Tổng kết, dặn dò, bài tập về nhà. Trong mỗi hoạt động đều có dự kiến thời gian hoàn thành.

▪ Từng nội dung của bài học đều được biên soạn song hành giữa 2 ngôn ngữ lập trình Python & C++.

+ **Phần 2. Luyện giải đề thi theo cấu trúc tỉnh:**

▪ Là bài giải **28** đề thi của Đà Nẵng (Cũ) qua các năm (Song hành giữa Python & C++), bao gồm:

◦ Các đề của Sở giáo dục và đào tạo Đà Nẵng như: Đề thi chọn học sinh giỏi lớp 9, Đề thi vào Trường THPT chuyên Lê Quý Đôn – Đà Nẵng.

◦ Các đề thi Tin học trẻ Bảng B (THCS) của Ban tổ chức hội thi Tin học trẻ Đà Nẵng.

▪ Mỗi câu trong từng đề thi đều được biên soạn theo cấu trúc gồm **4** phần như sau:

◦ Tóm tắt đề bài

◦ Ý tưởng giải

◦ Mô tả cụ thể các bước của thuật toán

◦ Code tham khảo (Python & C++): Từng dòng Code đều có chú thích bằng Tiếng Việt để người xem dễ dàng theo dõi nguyên lý hoạt động của các câu lệnh.

▪ Mỗi câu của từng đề thi đều có đủ **10** bộ Test, với các giá trị đầu vào tương ứng với tỉ lệ (%) của các ràng buộc (Substack) mà đề bài yêu cầu. Những bộ Test này đã được gửi ở tệp tin .RAR đính kèm cùng với tài liệu này.

▪ Cuối mỗi đề thi đều có Bảng kết quả chấm thử bằng phần mềm **Themis 1.9** với giới hạn thời gian chạy của mỗi bộ Test là **01** giây và giới hạn bộ nhớ là **1024** MiB.

Trên đây là những lời giới thiệu của tôi về bộ tài liệu này. Mặc dù bản thân tôi đã có nhiều cố gắng trong quá trình biên soạn, nhưng chắc chắn tài liệu sẽ không thể tránh khỏi những hạn chế và thiếu sót. Rất mong nhận được sự đóng góp ý kiến chân thành của mọi người để giúp cho bản thân tôi có thể rút ra được những kinh nghiệm quý báu, để những lần biên soạn tài liệu sau này của tôi được hoàn thiện hơn. Xin cảm ơn tất cả mọi người.

Hiện tại, tôi đã biên soạn xong **2** tập tài liệu dành cho luyện thi học sinh giỏi môn Tin 9 – THCS. **Tập 1** chính là bộ tài liệu này và **Tập 2** chính là bài giải của **32** đề thi qua các năm của các tỉnh: Quảng Nam (Cũ), Lào Cai (Cũ), Yên Bái (Cũ), Hòa Bình (Cũ), Bắc Giang (Cũ), Vĩnh Phúc (Cũ). Với cấu trúc bài giải của Tập 2 cũng giống y như cấu trúc của Tập 1 này. Hy vọng sẽ có được sự đón nhận của mọi người, mỗi sự đón nhận của mọi người chính là một nguồn động lực tiếp sức thêm cho tôi, để tôi tiếp tục **chia sẽ miễn phí** đến tất cả mọi người nội dung của **Tập 2**.

Trân trọng!

Đà Nẵng, tháng 12 năm 2025

Nguyễn Hữu Trung

MỤC LỤC

Chủ đề / Bài học	Trang
PHẦN 1. LỘ TRÌNH LUYỆN THI	6
Chủ đề 1. Nền tảng Python & C++	6
Bài 1. Cú pháp cơ bản, biến, kiểu dữ liệu	6
Bài 2. Cấu trúc rẽ nhánh	12
Bài 3. Cấu trúc lặp	19
Chủ đề 2. File I/O & Kỹ thuật lập trình cơ bản	25
Bài 4. Đọc/ghi file chuẩn (*.INP, *.OUT)	25
Bài 5. Hàm và tái sử dụng mã	32
Bài 6. Thuật toán cơ bản: Min/Max, Đếm, Tổng dãy	40
Chủ đề 3. Số học: Ước, Bội, Nguyên tố, Phân tích thừa số nguyên tố	46
Bài 7. Ước chung lớn nhất & Bội chung nhỏ nhất	46
Bài 8. Số nguyên tố - Sàng nguyên tố	53
Bài 9. Phân tích thừa số nguyên tố	62
Chủ đề 4. Số học nâng cao & Hình học cơ bản	72
Bài 10. Đồng dư thức	72
Bài 11. Hàm Phi Euler & Định lý Euler	82
Bài 12. Hình học cơ bản: Khoảng cách, Diện tích, Kiểm tra thẳng hàng	95
Chủ đề 5. Xử lý xâu cơ bản	105
Bài 13. Các thao tác cơ bản với xâu	105
Bài 14. Đếm từ & Đếm ký tự	113
Bài 15. Bài toán xâu con	121
Chủ đề 6. Xử lý xâu nâng cao	129
Bài 16. So khớp xâu cơ bản	129
Bài 17. Thuật toán KMP (Knuth – Morris – Pratt)	141
Bài 18. Mã hóa và Xử lý ký tự số	154
Chủ đề 7. Sắp xếp & Tìm kiếm	164
Bài 19. Các thuật toán sắp xếp cơ bản: Nổi bọt, Chọn, Chèn	164
Bài 20. Thuật toán Sắp xếp nhanh (Quick Sort)	177
Bài 21. Thuật toán Tìm kiếm nhị phân	189
Chủ đề 8. Cấu trúc dữ liệu	201
Bài 22. Ngăn xếp (Stack)	201
Bai 23. Hàng đợi (Queue)	217
Bài 24. Mảng cộng dồn	235
Chủ đề 9. Quy hoạch động (DP) cơ bản	252
Bài 25. Giới thiệu DP & Tìm dãy con liên tiếp có tổng lớn nhất (Kadane)	252
Bài 26. Tìm dãy con tăng dài nhất (LIS)	265
Bài 27. Giới thiệu bài toán DP dạng chuỗi - Dạng bảng 2 chiều	278

Chủ đề 10. Thuật toán quay lui	288
Bài 28. Sinh tổ hợp	288
Bài 29. Sinh hoán vị	298
Bài 30. Bài toán phân chia	308
Chủ đề 11. Tham lam & Chia để trị	321
Bài 31. Thuật toán tham lam: Bài toán Chọn việc	321
Bài 32. Thuật toán chia để trị: Sắp xếp trộn (Merge Sort)	332
Bài 33. Tham lam & Chia để trị: Bài toán tối ưu cơ bản	345
PHẦN 2. LUYỆN GIẢI ĐỀ THI THEO CẤU TRÚC TỈNH	357
Đề số 1. Học sinh giỏi 9 – Đà Nẵng – Năm học: 2024 – 2025	357
Đề số 2. Học sinh giỏi 9 – Đà Nẵng – Năm học: 2023 – 2024	373
Đề số 3. Học sinh giỏi 9 – Đà Nẵng – Năm học: 2022 – 2023	387
Đề số 4. Tin học trẻ Bảng B (THCS) – Đà Nẵng – Năm học: 2022 – 2023	399
Đề số 5. THPT Chuyên Lê Quý Đôn – Đà Nẵng – Năm học: 2022 – 2023	419
Đề số 6. Học sinh giỏi 9 – Đà Nẵng – Năm học: 2021 – 2022	432
Đề số 7. Tin học trẻ Bảng B (THCS) – Đà Nẵng – Năm học: 2021 – 2022	442
Đề số 8. THPT Chuyên Lê Quý Đôn – Đà Nẵng – Năm học: 2021 – 2022	456
Đề số 9. Học sinh giỏi 9 – Đà Nẵng – Năm học: 2020 – 2021	472
Đề số 10. Tin học trẻ Bảng B (THCS) – Đà Nẵng – Năm học: 2020 – 2021	485
Đề số 11. Học sinh giỏi 9 – Đà Nẵng – Năm: 2020	506
Đề số 12. Tin học trẻ Bảng B (THCS) – Đà Nẵng – Năm: 2020	516
Đề số 13. THPT Chuyên Lê Quý Đôn – Đà Nẵng – Năm: 2020	527
Đề số 14. Học sinh giỏi 9 – Đà Nẵng – Năm học: 2018 – 2019	536
Đề số 15. Tin học trẻ Bảng B (THCS) – Đà Nẵng – Năm học: 2018 – 2019	546
Đề số 16. THPT Chuyên Lê Quý Đôn – Đà Nẵng – Năm: 2019	559
Đề số 17. Học sinh giỏi 9 – Đà Nẵng – Năm học: 2017 – 2018	568
Đề số 18. Tin học trẻ Bảng B (THCS) – Đà Nẵng – Năm học: 2017 – 2018	579
Đề số 19. THPT Chuyên Lê Quý Đôn – Đà Nẵng – Năm: 2018	591
Đề số 20. Học sinh giỏi 9 – Đà Nẵng – Năm học: 2016 – 2017	605
Đề số 21. Tin học trẻ Bảng B (THCS) – Đà Nẵng – Năm học: 2016 – 2017	617
Đề số 22. THPT Chuyên Lê Quý Đôn – Đà Nẵng – Năm: 2017	631
Đề số 23. Học sinh giỏi 9 – Đà Nẵng – Năm học: 2015 – 2016	648
Đề số 24. THPT Chuyên Lê Quý Đôn – Đà Nẵng – Năm: 2016	662
Đề số 25. Học sinh giỏi 9 – Đà Nẵng – Năm học: 2014 – 2015	673
Đề số 26. THPT Chuyên Lê Quý Đôn – Đà Nẵng – Năm: 2015	687
Đề số 27. Học sinh giỏi 9 – Đà Nẵng – Năm học: 2012 – 2013	699
Đề số 28. THPT Chuyên Lê Quý Đôn – Đà Nẵng – Năm: 2013	715

PHẦN 1. LỘ TRÌNH LUYỆN THI

CHỦ ĐỀ 1. NỀN TẢNG PYTHON & C++

BÀI 1. CÚ PHÁP CƠ BẢN, BIẾN, KIỂU DỮ LIỆU

I. Mục Tiêu:

- **Kiến thức:** Học sinh (**HS**) nắm vững khái niệm **biến**, **kiểu dữ liệu cơ bản** (int, float, str/string), và **toán tử số học**.
- **Kỹ năng:**
  + HS thành thạo thao tác **Nhập/Xuất Console** (input()/print() trong Python; cin/cout trong C++).
  + HS hiểu rõ và thực hiện thành thạo thao tác **Ép kiểu** (**Type Casting**), đặc biệt khi nhận dữ liệu từ đầu vào.
- **Tư duy:** HS bước đầu nhận diện được lỗi sai định dạng nhập và tầm quan trọng của việc kiểm soát kiểu dữ liệu.

II. Phân Bổ Thời Gian (90 Phút):

Hoạt Động	Nội Dung	Thời gian (Phút)
Khởi động	Giới thiệu cách thức ôn luyện hiệu quả	10
Lý thuyết 1	Khái Niệm Cơ Bản: Biến, Kiểu Dữ Liệu và Toán Tử	15
Lý thuyết 2	Nhập, Xuất và Ép Kiểu Console	20
Thực hành 1	Bài 1. Tính tổng – hiệu – tích – thương của hai số	20
Thực hành 2	Bài 2. Xử lý sai định dạng nhập	15
Tổng kết	Bài tập về nhà, Củng cố	10
Tổng cộng		**90**

III. Nội Dung Giảng Dạy Chi Tiết & Demo:

1. Khái Niệm Cơ Bản: Biến, Kiểu Dữ Liệu và Toán Tử (15 phút)

Khái Niệm	Mô Tả	Python	C++
Biến	Vùng nhớ để lưu trữ dữ liệu.	Tự động xác định kiểu (Dynamically Typed).	**Phải khai báo kiểu** trước khi dùng (Statically Typed).
Kiểu Dữ Liệu	int (nguyên), float/double (thực), str/string (chuỗi).		**Lưu ý:** C++ có int (4 bytes), long long (8 bytes), float (4 bytes), double (8 bytes).
Toán tử	+, -, *, / (chia), % (chia dư).	5 / 2 = 2.5	5 / 2 = 2 (phải ép kiểu để ra 2.5)

2. Nhập, Xuất và Ép Kiểu Console (20 phút)

2.1. Cấu trúc:

2.1.1. Python: Viết code thẳng vào giao diện giống như soạn thảo văn bản trong Word

2.1.2. C++: Phải viết vào trong khung bắt buộc. Các thư viện được khai báo tùy thuộc vào từng chương trình, nhưng phía dưới là việc khai báo các thư viện theo đề xuất của tôi.

```
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
using namespace std;
int main()
{
    // Các dòng code được viết tại đây
    return 0;
}
```

2.2. Nhập dữ liệu (Từ bàn phím):

2.2.1. Python:

- Hàm input() luôn trả về chuỗi (str).
- Code Python:

```
ten = input("Nhap ten: ")   # Vẫn là chuỗi "10"
so_a_str = input("Nhap so a: ")
```

2.2.2. C++:

- Đối tượng **cin** và toán tử >>. **Đọc trực tiếp vào biến số** nếu đã khai báo kiểu.
- Code C++:

```
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
using namespace std;
int main()
{
    int a;
    cout << "Nhap a:";
    cin >> a;   // a là số nguyên
    return 0;
}
```

2.3. Xuất dữ liệu (Từ bàn phím):

2.3.1. Python:

- Sử dụng hàm print()
- Code Python:

```
ten = "Nguyen Van A"
print("Xin chao!", ten)
```

=> Kết quả xuất ra màn hình là:

```
Xin chao! Nguyen Van A
```

2.3.2. C++:

- Sử dụng đối tượng cout và toán tử <<.
- Code C++:

```cpp
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
using namespace std;
int main()
{
    string ten = "Nguyen Van A";
    cout << "Xin chao! " << ten << endl;
    // Kí hiệu endl có nghĩa là: Sau khi in ra xong thì xuống dòng
    return 0;
}
```

=> Kết quả xuất ra màn hình là:

```
Xin chao! Nguyen Van A
```

2.4. Ép kiểu:

2.4.1. Python:

- Bắt buộc chuyển từ str sang số trước khi tính toán.
- Code Python:

```python
# Nhập số a
so_a_str = input("Nhap so a: ")
# Chuyển chuỗi thành số nguyên
so_a = int(so_a_str)
# Chuyển chuỗi thành số thực
so_pi = float("3.14")
```

2.4.2. C++:

- Dùng để chuyển giữa các kiểu số để lấy kết quả chính xác (ví dụ: chia).
- Code C++:

```cpp
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
using namespace std;
int main()
{
    int a = 10;
    // Ép kiểu a thành float (số thực)
    float b = float(a);
    return 0;
}
```

IV. Thực Hành Kèm Lời Giải Chi Tiết:

1. Bài 1. Tính tổng – hiệu – tích – thương của hai số. (20 phút)

- **Yêu cầu:** Nhập 2 số A, B từ Console. In ra màn hình Tổng, Hiệu, Tích, và Thương. **Thương phải là số thực**.

1.1. Python:

- Code Python:

```python
# 1. Nhập dữ liệu và ép kiểu ngay sang float để tính toán chính xác
A = float(input('Nhap A: '))
B = float(input('Nhap B: '))
# 2. Xuất kết quả với f-string
print(f'Tong: {A + B}')
print(f'Hieu: {A - B}')
print(f'Tich: {A * B}')
print(f'Thuong: {A / B}')   # Phep chia luon ra float
```

=> Kết quả khi chạy xong chương trình:

```
Nhap A: 5
Nhap B: 3
Tong: 8.0
Hieu: 2.0
Tich: 15.0
Thuong: 1.6666666666666667
```

- **Nhận xét:** Phải ép kiểu bằng float() vì input() trả về chuỗi. Sử dụng f-string (f' ') giúp định dạng kết quả dễ đọc.

1.2. C++:

- Code C++:

```cpp
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
using namespace std;
int main()
{
    int a, b;
    cout << "Nhap A va B: ";
    cin >> a >> b;
    cout << "Tong: " << a + b << endl;
    cout << "Hieu: " << a - b << endl;
    cout << "Tich: " << a * b << endl;
    // Ép kiểu một trong hai số hạng thành double để phép chia là số thực
    cout << "Thuong: " << (double)a / b << endl;
    return 0;
}
```

=> Kết quả khi chạy xong chương trình:

```
Nhap A va B: 5 3
Tong: 8
Hieu: 2
Tich: 15
Thuong: 1.66667
```

- Nhận xét: Khai báo int cho a, b. Khi tính thương, (double)a / b là ép kiểu a thành double để C++ thực hiện phép chia ra số thực thay vì chia lấy phần nguyên.

2. Bài 2. Xử lý sai định dạng nhập. (15 phút)

- **Hoạt động:** Yêu cầu HS chạy lại Bài 1 và cố ý nhập **"abc"** thay vì số.
- **Mục đích:** Nhận thức về sự cố khi kiểu dữ liệu không khớp.

2.1. Python:

- Nhận xét: Báo lỗi ValueError (Lỗi giá trị).

- Giải thích: Chương trình không thể chuyển chuỗi "abc" thành số (int hoặc float) khi gặp lệnh int() hoặc float().

2.2. C++:

- Kết quả khi chạy chương trình:

```
Nhap A va B: abc 3
Tong: 406614080
Hieu: -406614080
Tich: 0
Thuong: 0
```

- Nhận xét: Biến a và b có thể nhận giá trị 0 hoặc giá trị rác. Chương trình tiếp tục chạy nhưng kết quả sai.
- Giải thích: cin bị lỗi khi gặp ký tự không phải số. Luồng nhập lỗi khiến việc gán giá trị thất bại nhưng không dừng chương trình.

2.3. Chốt kiến thức: Đây là **lỗi runtime** do đầu vào không hợp lệ. Để giải quyết, chúng ta cần sử dụng các cấu trúc **kiểm soát lỗi** (như if/else hoặc try/except), sẽ được học trong các buổi sau.

V. Bài Tập Về Nhà Kèm Lời Giải Tham Khảo:

1. Bài 1. Cho 3 số, tìm số lớn nhất.

1.1. Python:

- Code Python:

```python
A = int(input('A: '))
B = int(input('B: '))
C = int(input('C: '))
# Dùng hàm max có sẵn
max_val = max(A, B, C)
print(f'So lon nhat: {max_val}')
```

=> Kết quả khi chạy chương trình:

```
A: 5
B: 9
C: 3
So lon nhat: 9
```

- Nhận xét: Python cung cấp hàm max() tích hợp sẵn, giúp việc tìm giá trị lớn nhất trở nên rất ngắn gọn và hiệu quả.

1.2. C++:

- Code C++:

```cpp
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
using namespace std;
int main()
{
    int a, b, c;
    cout << "Nhap 3 so: ";
    cin >> a >> b >> c;
    // Dùng hàm max trong thư viện tổng hợp <bits/stdc++.h>
    int max_val = max({a, b, c});
    cout << "So lon nhat: " << max_val << endl;
    return 0;
}
```

=> Kết quả khi chạy chương trình:

```
Nhap 3 so: 5 9 3
So lon nhat: 9
```

- Nhận xét: C++ cũng có hàm max() trong thư viện tổng hợp <bits/stdc++.h>. Lưu ý rằng đối với 3 số trở lên, cú pháp max({a, b, c})là tiện lợi nhất.

2. Bài 2. Viết chương trình đọc file A.INP ghi B.OUT với kết quả a + b.

- **Mục đích:** Bắt đầu làm quen với I/O File, kỹ thuật cốt lõi trong lập trình thi đấu.

- **Giải thích đề bài:**

+ Viết chương trình nhập vào **2** số nguyên **a, b** từ file **A.INP**

+ Yêu cầu: Hãy ghi ra file **B.OUT** tổng của **2** số nguyên **a, b** vừa nhập ở trên.

+ Dữ liệu vào: Đọc vào file văn bản **A.INP** hai số nguyên **a, b** ở trên cùng 1 dòng, mỗi số cách nhau một khoảng trắng.

+ Dữ liệu ra: Ghi ra file văn bản **B.OUT** gồm chỉ 1 dòng ghi 1 số nguyên là kết quả cần tìm

+ Ví dụ:

A.INP	B.OUT
5 3	8

2.1. Python:

- **Code Python:**

```python
# Đọc dữ liệu từ file A.INP
fi = open('A.INP', 'r')      # Mở file A.INP để đọc, nhớ là phải có 'r'
data = fi.read().split()     # Đọc tất cả dữ liệu từ file A.INP
fi.close()                   # Đóng file A.INP sau khi đã đọc xong dữ liệu
# Tách dữ liệu trong file A.INP vừa đọc thành từng chuỗi a, b riêng lẻ
# Phải dùng int() để ép kiểu trước khi cộng: a = int(data[0]), b = int(data[1])
a = int(data[0])
b = int(data[1])
tong = a + b                 # Tính tổng của 2 số nguyên a, b
# Ghi kết quả ra file B.OUT
fo = open('B.OUT', 'w')      # Mở file B.OUT để ghi kết quả, nhớ là phải có 'w'
fo.write(f'{tong}')          # Ghi kết quả ra file B.OUT
fo.close()                   # Đóng file B.OUT sau khi đã ghi xong kết quả
```

- **Nhận xét:** Đọc tất cả dữ liệu từ file **A.INP** và tách chúng thành các chuỗi. Phải dùng int() để ép kiểu trước khi cộng, và ghi kết quả ra file **B.OUT**.

2.2. C++:

- **Code C++:**

```cpp
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
#include <fstream>           // Thư viện đọc, ghi file
using namespace std;
int main()
{
    // Đọc dữ liệu từ file A.INP
    ifstream fi("A.INP");    // Mở file A.INP để đọc
    int a, b;
    fi >> a >> b;            // Đọc 2 số a, b từ file A.INP
    fi.close();              // Đóng file A.INP sau khi đọc xong
    int tong = a + b;        // Tính tổng của 2 số nguyên a, b
    ofstream fo("B.OUT");    // Mở file B.OUT để ghi kết quả
    fo << tong;              // Ghi kết quả ra file B.OUT
    fo.close();              // Đóng file B.OUT sau khi đã ghi xong kết quả
    return 0;
}
```

- **Nhận xét:** Cần thư viện <fstream>. ifstream và ofstream hoạt động tương tự cin/cout. C++ tự động ép kiểu dữ liệu đọc được thành int vì a, b đã được khai báo là int.

VI. Tổng Kết & Dặn Dò:

- **Củng cố chính:** Luôn nhớ sự khác biệt lớn nhất giữa hai ngôn ngữ là **quy tắc ép kiểu** khi nhận đầu vào và khi thực hiện phép chia số thực.
- **Dặn dò:** Hoàn thành 2 bài tập về nhà và chuẩn bị các câu hỏi cho buổi tiếp theo về cấu trúc điều kiện (if/else).

BÀI 2. CẤU TRÚC RẼ NHÁNH

I. Mục Tiêu:

- **Kiến thức:** Học sinh (**HS**) nắm vững các **toán tử so sánh** và **biểu thức logic** (AND, OR, NOT).
- **Kỹ năng:** HS thành thạo cách sử dụng cấu trúc **if/elif/else** trong Python và **if/else** cùng **switch/case** trong C++.
- **Tư duy:** HS biết cách phân tích và mô hình hóa các bài toán có nhiều điều kiện (Phân loại tam giác, Kiểm tra năm nhuận) thành các cấu trúc rẽ nhánh lồng nhau hoặc liên tiếp.

II. Phân Bổ Thời Gian (90 Phút):

Hoạt Động	Nội Dung	Thời gian (Phút)
Khởi động	Kiểm tra bài cũ (Bài tập File I/O)	10
Lý thuyết 1	Toán tử So sánh & Logic	10
Lý thuyết 2	Cấu trúc if/else cơ bản & switch	15
Thực hành 1	Bài 1. Phân loại tam giác	25
Thực hành 2	Bài 2. Kiểm tra năm nhuận	15
Tổng kết	Bài tập về nhà, Củng cố	15
Tổng cộng		**90**

III. Nội Dung Giảng Dạy Chi Tiết & Demo:

1. So Sánh và Biểu Thức Logic (10 phút):

Toán tử	Ý nghĩa	Python	C++
Bằng nhau		==	==
Khác nhau		!=	!=
Lớn hơn/Bé hơn		>, <	>, <
Lớn hơn/Bé hơn hoặc bằng		>=, <=	>=, <=
Và (AND)	Cả hai điều kiện đều TRUE	and	**&&**
Hoặc (OR)	Một trong hai điều kiện TRUE	or	**||**
Không (NOT)	Phủ định điều kiện	not	**!**

- Ví dụ về Biểu thức Logic: Điều kiện để x nằm ngoài đoạn [10, 20]:
  + Python: x < 10 or x > 20
  + C++: (x < 10) || (x > 20)

2. Cấu Trúc Rẽ Nhánh Cơ Bản (15 phút):

2.1. Python:

2.1.1. Cấu trúc cơ bản (if):

- Code Python:

```python
if a > b:
    a = a + 3
    print(a)
```

- Giải thích: Cú pháp dùng dấu hai chấm (:) và thụt lề để xác định khối lệnh.

2.1.2. Cấu trúc nhiều điều kiện (if/elif/else):

- Code Python:

```python
diem = 8
if diem >= 9:
    print("Loai A")
elif diem >= 7:
    print("Loai B")
else:
    print("Loai C")
```

2.1.3. Cấu trúc rẽ nhánh cố định: Không có **switch** riêng, dùng **if/elif** thay thế

2.2. C++:

2.2.1. Cấu trúc cơ bản (if):

- Code C++:

```cpp
if (a > b)
{
    a = a + 3;
    cout << a;
}
```

- Giải thích: Phải có dấu mở ngoặc và đóng ngoặc đơn "()" để kiểm tra điều kiện. Cú pháp dùng cặp dấu ngoặc nhọn "{}" để xác định khối lệnh.

2.1.2. Cấu trúc nhiều điều kiện (if/else if/else):

- Code C++:

```
int diem = 8;
if (diem >= 9)
{
    cout << "A";
}
else if (diem >= 7)
{
    cout << "B";
}
else
{
    cout << "C";
}
```

2.1.3. Cấu trúc rẽ nhánh cố định (switch/case): Dùng khi kiểm tra một biến có giá trị bằng **case** nào đó.

2.1.4. Switch:

- Code C++:

```
int choice = 5;
switch (choice)
{
    case 1:
        cout << "Chon 1";
        break;  // Break; => Rất quan trọng bắt buộc phải có
    case 2:
        cout << "Chon 2";
        break;  // Break; => Rất quan trọng bắt buộc phải có
    default:
        cout << "Khac";
}
```

- Nhận xét: Lệnh **break;** là **bắt buộc** phải có để thoát khỏi switch sau khi case thực hiện lệnh.

IV. Thực Hành Kèm Code Tham Khảo:

1. Bài 1. Phân loại tam giác. (25 phút)

- **Yêu cầu:** Nhập 3 cạnh a, b, c. Kiểm tra và in ra loại tam giác (Đều, Cân, Vuông, Thường, hoặc Không phải tam giác).

- **Điều kiện cần kiểm tra trước:** a +b > c và a + c > b và b + c > a (Không phải tam giác nếu điều kiện này FALSE).

1.1. Code Python:

```python
# Nhập vào 3 cạnh a, b, c
a, b, c = map(float, input("Nhap 3 canh: ").split())
# Kiểm tra điều kiện tam giác
if a + b > c and a + c > b and b + c > a:
    if a == b and b == c:
        print("Tam giac Deu")
    elif a == b or a == c or b == c:
        print("Tam giac Can")
    elif a*a + b*b == c*c or a*a + c*c == b*b or b*b + c*c == a*a:
        print("Tam giac Vuong")
    else:
        print("Tam giac Thuong")
else:
    print("Khong phai tam giac")
```

=> Kết quả sau khi chạy chương trình:

```
Nhap 3 canh: 3 4 5
Tam giac Vuong
```

1.2. Code C++:

```cpp
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
using namespace std;
int main()
{
    // Nhập vào 3 cạnh a, b, c
    float a, b, c;
    cout << "Nhap 3 canh: ";
    cin >> a >> b >> c;
    // Kiểm tra điều kiện tam giác
    if (a + b > c && a + c > b && b + c > a)
    {
        if (a == b && b == c)
            cout << "Tam giac Deu";
        else if (a == b || a == c || b == c)
            cout << "Tam giac Can";
        else if (a*a + b*b == c*c || a*a + c*c == b*b || b*b + c*c == a*a)
            cout << "Tam giac Vuong";
        else
            cout << "Tam giac Thuong";
    }
    else
        cout << "Khong phai tam giac";
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
Nhap 3 canh: 3 4 5
Tam giac Vuong
```

2. Bài 2. Kiểm tra năm nhuận. (15 phút)

- **Yêu cầu:** Nhập một năm. Kiểm tra và in ra đó có phải là năm nhuận hay không.
- **Quy tắc:** Năm nhuận là năm: **Chia hết cho 4** và (KHÔNG chia hết cho 100) hoặc (**Chia hết cho 400**).

2.1. Code Python:

```python
# Nhập vào năm để kiểm tra
nam = int(input('Nhap nam: '))
# Kiểm tra theo quy tắc năm nhuận
if (nam % 4 == 0 and nam % 100 != 0) or (nam % 400 == 0):
    print(f'Nam {nam} la nam nhuan.')
else:
    print(f'Nam {nam} KHONG phai nam nhuan.')
```

=> Kết quả sau khi chạy chương trình:

```
Nhap nam: 2025
Nam 2025 KHONG phai nam nhuan.
```

2.2. Code C++:

```cpp
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
using namespace std;
int main()
{
    // Nhập vào năm để kiểm tra
    int nam;
    cout << "Nhap nam: ";
    cin >> nam;
    // Kiểm tra theo quy tắc năm nhuận
    if ((nam % 4 == 0 && nam % 100 != 0) || (nam % 400 == 0))
    {
        cout << "Nam " << nam << " la nam nhuan.";
    }
    else
    {
        cout << "Nam " << nam << " KHONG phai nam nhuan.";
    }
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
Nhap nam: 2025
Nam 2025 KHONG phai nam nhuan.
```

V. Bài Tập Về Nhà Kèm Gợi Ý Code:

1. Bài 1. Tính tiền điện theo bậc.

- **Mô tả:** Nhập số điện tiêu thụ (kW). Tính tiền điện theo công thức:
  + Bậc 1 (0 - 50k): 1678 VNĐ/kW
  + Bậc 2 (51 - 100k): 1734 VNĐ/kW
  + Bậc 3 (101k trở lên): 2014 VNĐ/kW

1.1. Code Python:

```python
# Nhập vào số kW điện
so_kW = int(input('Nhap so kW dien: '))
# Tính số tiền điện theo bậc
tien = 0
if so_kW <= 50:
    tien = so_kW* 1678
elif so_kW <= 100:  # Từ 51 đến 100
    tien = 50 * 1678 + (so_kW - 50) * 1734
else:               # Trên 100
    tien = 50 * 1678 + 50 * 1734 + (so_kW - 100) * 2014
# In kết quả
print(f'Tong tien dien: {tien} VND')
```

=> Kết quả sau khi chạy chương trình:

```
Nhap so kW dien: 120
Tong tien dien: 210880 VND
```

1.2. Code C++:

```cpp
#include <iostream>         // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>    // Thư viện tổng hợp
using namespace std;
int main()
{
    // Nhập vào số kW điện
    int so_kW;
    cout << "Nhap so kW dien: ";
    cin >> so_kW;
    // Tương tự Python, tính tiền theo từng bậc
    long long tien = 0;     // Dùng long long để tránh tràn số
    if (so_kW <= 50)
    {
        tien = so_kW * 1678;
    }
    else if (so_kW <= 100)  // Từ 51 đến 100
    {
        tien = 50 * 1678 + (so_kW - 50) * 1734;
    }
    else                    // Trên 100
    {
        tien = 50 * 1678 + 50 * 1734 + (so_kW - 100) * 2014;
    }
    // In kết quả
    cout << "Tong tien dien: " << tien << " VND" << endl;
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
Nhap so kW dien: 120
Tong tien dien: 210880 VND
```

2. Bài 2. Bài phân loại học lực dựa trên điểm.

- **Mô tả:** Nhập điểm trung bình (D). Phân loại học lực:
  + $D \geq 9.0$: Xuất sắc
  + $8.0 \leq D < 9.0$: Giỏi
  + $6.5 \leq D < 8.0$: Khá

+ 5.0 ≤ D < 6.5: Trung bình
+ D < 5.0: Yếu

2.1. Code Python:

```python
diem = float(input("Nhap diem TB: "))
if diem >= 9.0:
    print("Xuat sac")
elif diem >= 8.0:
    print("Gioi")
elif diem >= 6.5:
    print("Kha")
elif diem >= 5.0:
    print("Trung binh")
else:
    print("Yeu")
```

=> Kết quả sau khi chạy chương trình:

```
Nhap diem TB: 8.5
Gioi
```

2.2. Code C++:

```cpp
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
using namespace std;
int main()
{
    float diem;
    cout << "Nhap diem TB: ";
    cin >> diem;
    if (diem >= 9.0)
    {
        cout << "Xuat sac" << endl;
    }
    else if (diem >= 8.0)
    {
        cout << "Gioi" << endl;
    }
    else if (diem >= 6.5)
    {
        cout << "Kha" << endl;
    }
    else if (diem >= 5.0)
    {
        cout << "Trung binh" << endl;
    }
    else
    {
        cout << "Yeu" << endl;
    }
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
Nhap diem TB: 8.5
Gioi
```

VI. Tổng Kết & Dặn Dò:

- **Củng cố chính:** Luôn nhớ nguyên tắc của **if/elif/else** hoặc **if/else if/else**: **Chỉ một khối lệnh được thực hiện**. Nếu điều kiện **TRUE** ở **if** đầu tiên, các **elif** (hoặc **else if**) sau đó sẽ bị bỏ qua.
- **Dặn dò:** Hoàn thành bài tập về nhà và chuẩn bị trước kiến thức về **Cấu trúc lặp** (**for/while**) cho buổi tiếp theo.

BÀI 3. CẤU TRÚC LẶP

I. Mục Tiêu:

- **Kiến thức:** Học sinh (**HS**) nắm vững hai loại cấu trúc lặp cơ bản: **lặp xác định** (**for**) và **lặp không xác định** (**while**).
- **Kỹ năng:** HS thành thạo việc sử dụng **for** và **while** để giải quyết các bài toán tính tổng, in dãy số, và sử dụng **vòng lặp lồng nhau** (Nested Loops).
- **Tư duy:** HS biết cách chọn cấu trúc lặp phù hợp cho từng bài toán cụ thể và tối ưu hóa điều kiện dừng.

II. Phân Bổ Thời Gian (90 Phút):

Hoạt Động	Nội Dung	Thời gian (Phút)
Khởi động	Kiểm tra bài cũ (if/else)	10
Lý thuyết 1	Giới thiệu Vòng lặp (for và while)	10
Lý thuyết 2	Vòng lặp lồng nhau & break/continue	10
Thực hành 1	Bài 1. In bảng cửu chương (Lồng nhau)	25
Thực hành 2	Bài 2. Tính tổng S = 1 + 1/2 + ... + 1/n	20
Tổng kết	Bài tập về nhà, Củng cố	15
Tổng cộng		**90**

III. Nội Dung Giảng Dạy Chi Tiết & Demo:

1. Lý Thuyết: for và while (20 phút):

1.1. Cấu trúc vòng lặp for:

- **Khái niệm và ứng dụng:** Lặp xác định (Biết trước số lần lặp). Thường dùng để duyệt qua các phần tử của một tập hợp (chuỗi, danh sách) hoặc lặp một số lần cố định.
- **Python:**

  Duyệt qua **range** hoặc tập hợp.
- **C++:**

  Cần 3 thành phần: Khởi tạo, Điều kiện, Tăng/Giảm bước.

1.2. Code mẫu for:

- **Ví dụ:** In từ 1 đến 5
- **Python:**

```python
for i in range(1, 6):
    print(i)
```

- **C++:**

```
for (int i = 1; i <= 5; i++)
{
    cout << i << " ";
}
```

1.3. Vòng lặp while:

- **Khái niệm và ứng dụng: Lặp không xác định** (Lặp cho đến khi điều kiện FALSE). Thường dùng cho các bài toán cần điều kiện dừng phức tạp (ví dụ: tìm kiếm, xử lý input).
- **Python:** Lặp cho đến khi điều kiện FALSE.
- **C++:** Lặp cho đến khi điều kiện FALSE.

1.4. Code mẫu while:

- **Ví dụ:** Lặp khi x < 10
- **Python:**

```
x = 1
while x < 10:
    print(x)
    x = x + 1   # Tăng biến đếm lên 1 đơn vị
```

- **C++:**

```
int x = 1;
while (x < 10)
{
    cout << x << " ";
    x = x + 1;  // Tăng biến đếm lên 1 đơn vị
}
```

1.5. Vòng lặp lồng nhau:

Một vòng lặp nằm bên trong thân của vòng lặp khác. Dùng để xử lý ma trận, tọa độ hoặc in hình.

1.6. break:

Thoát khỏi vòng lặp gần nhất ngay lập tức.

1.7. continue:

Bỏ qua các lệnh còn lại trong lần lặp hiện tại và chuyển sang lần lặp tiếp theo.

IV. Thực Hành Kèm Lời Giải Tham Khảo:

1. Bài 1. In bảng cửu chương. (25 phút)

- **Yêu cầu:** In ra bảng cửu chương từ 1 x 1 đến 9 x 9. (Sử dụng vòng lặp lồng nhau)
- **Logic:**
  + Vòng lặp ngoài **i**: Chạy từ 1 đến 9 (là số nhân).
  + Vòng lặp trong **j**: Chạy từ 1 đến 9 (là thừa số).
  + Công thức: **i x j = Kết quả**.

1.1. Code Python:

Code Python	Kết quả sau khi chạy chương trình

`# Vong lap ngoai (i chay tu 1 den 9)`
`for i in range(1, 10):`
`  print(f"--- Bang nhan {i} ---")`
`  # Vong lap trong (j chay tu 1 den 9)`
`  for j in range(1, 10):`
`  ket_qua = i * j`
`  print(f"{i} x {j} = {ket_qua}")`
	`--- Bang nhan 1 ---`
	`1 x 1 = 1`
	`1 x 2 = 2`
	`1 x 3 = 3`
	`1 x 4 = 4`
	`1 x 5 = 5`
	`1 x 6 = 6`
	`1 x 7 = 7`
	`1 x 8 = 8`
	`1 x 9 = 9`
	`--- Bang nhan 2 ---`
	`2 x 1 = 2`
	`2 x 2 = 4`
	`2 x 3 = 6`
	`2 x 4 = 8`
	`2 x 5 = 10`
	`2 x 6 = 12`
	`2 x 7 = 14`
	`2 x 8 = 16`
	`2 x 9 = 18`
	`--- Bang nhan 3 ---`
	`3 x 1 = 3`
	`3 x 2 = 6`
	`3 x 3 = 9`

- **Giải thích Code Python:** Dùng **range(1, 10)** để lặp từ 1 đến 9. **f-string** giúp định dạng kết quả nhanh và đẹp.

1.2. Code C++:

Code C++	Kết quả sau khi chạy chương trình

`#include <iostream>  // Thư viện nhập, xuất dữ liệu`
`#include <bits/stdc++.h>  // Thư viện tổng hợp`
`using namespace std;`
`int main()`
`{`
`  // Vong lap ngoai (i chay tu 1 den 9)`
`  for (int i = 1; i <= 9; i++)`
`  {`
`  cout << "--- Bang nhan " << i << " ---" << endl;`
`  // Vong lap trong (j chay tu 1 den 9)`
`  for (int j = 1; j <= 9; j++)`
`  {`
`  int ket_qua = i * j;`
`  cout << i << " x " << j << " = " << ket_qua << endl;`
`  }`
`  }`
`  return 0;`
`}`
	`--- Bang nhan 1 ---`
	`1 x 1 = 1`
	`1 x 2 = 2`
	`1 x 3 = 3`
	`1 x 4 = 4`
	`1 x 5 = 5`
	`1 x 6 = 6`
	`1 x 7 = 7`
	`1 x 8 = 8`
	`1 x 9 = 9`
	`--- Bang nhan 2 ---`
	`2 x 1 = 2`
	`2 x 2 = 4`
	`2 x 3 = 6`
	`2 x 4 = 8`
	`2 x 5 = 10`
	`2 x 6 = 12`
	`2 x 7 = 14`
	`2 x 8 = 16`
	`2 x 9 = 18`
	`--- Bang nhan 3 ---`
	`3 x 1 = 3`
	`3 x 2 = 6`
	`3 x 3 = 9`

- **Giải thích Code C++:** Cú pháp **for (Khởi tạo; Điều kiện; Bước nhảy)** là bắt buộc. Sử dụng **endl** để xuống dòng sau mỗi phép tính.

2. Bài 2. Tính tổng S = 1 + 1/2 + … + 1/n. (20 phút)

- **Yêu cầu:** Nhập số nguyên dương **n**. Tính tổng **S** là tổng của dãy phân số.
- **Logic:** Cần dùng biến **số thực** (float/double) để lưu tổng **S** và kết quả phép chia **1/i**.

2.1. Code Python:

```python
n = int(input("Nhap N: "))
S = 0.0     # Khởi tạo tổng là số thực
for i in range(1, n + 1):
    S = S + 1 / i
print(f"Tong S = {S:.4f}")  # Làm tròn 4 chữ số thập phân
```

=> Kết quả sau khi chạy chương trình:

```
Nhap N: 10
Tong S = 2.9290
```

- **Giải thích Code Python:** Khai báo **S = 0.0** là **float**. Trong Python, phép chia **1 / i** luôn trả về **float**, nên kết quả chính xác.

2.1. Code C++:

```cpp
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
using namespace std;
int main()
{
    int n;
    cout << "Nhap N: ";
    cin >> n;
    double S = 0.0; // Bắt buộc phải là double hoặc float
    for (int i = 1; i <= n; i++)
    {
        // Bắt buộc ép kiểu 1/i thành số thực S = S + 1.0 / i;
        // Hoặc dùng: S = S + (double)1 / i;
        S = S + (double)1 / i;
    }
    // In kết quả với 4 chữ số thập phân
    cout << fixed << setprecision(4) << "Tong S = " << S << endl;
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
Nhap N: 10
Tong S = 2.9290
```

- **Giải thích Code C++: Bắt buộc** phải dùng **double** hoặc **float** cho **S**. Phép tính **double(1) / i** đảm bảo phép chia là **chia số thực**, nếu không **1 / i** sẽ là chia số nguyên (kết quả bằng **0** với **i > 1**).

V. Bài Tập Về Nhà Kèm Code Tham Khảo:

1. Bài 1. Kiểm tra số nguyên tố.

- **Mô tả:** Nhập số nguyên dương N. Kiểm tra xem N có phải là số nguyên tố hay không.

- **Logic:** Số nguyên tố là số chỉ chia hết cho 1 và chính nó. Cần dùng vòng lặp (for hoặc while) để kiểm tra xem N có chia hết cho bất kỳ số nào từ 2 đến căn bậc 2 của N (sqrt(N)) hay không. Dùng break nếu tìm thấy ước.

1.1. Code Python (Dùng for và break):

Code Python	Kết quả sau khi chạy chương trình

`n = int(input("Nhap N: "))`
`nguyento = True`
`if n < 2:`
`  nguyento = False`
`else:`
`  # Kiểm tra từ 2 đến căn bậc hai của n`
`  for i in range(2, int(n**0.5) + 1):`
`  if n % i == 0:`
`  nguyento = False`
`  break  # Thoát ngay khi tìm thấy ước`
`  if nguyento:`
`  print(f"{n} la so nguyen to")`
`  else:`
`  print(f"{n} KHONG la so nguyen to")`
	- Test 1:
	`Nhap N: 29`
	`29 la so nguyen to`
	- Test 2:
	`Nhap N: 10`
	`10 KHONG la so nguyen to`

1.2. Code C++ (Dùng for và break):

Code Python	Kết quả sau khi chạy chương trình

`#include <iostream>  // Thư viện nhập, xuất dữ liệu`
`#include <bits/stdc++.h>  // Thư viện tổng hợp`
`using namespace std;`
`int main()`
`{`
`  int n;`
`  cout << "Nhap N: ";`
`  cin >> n;`
`  bool nguyento = true;`
`  if (n < 2)`
`  {`
`  nguyento = false;`
`  }`
`  else`
`  // Kiểm tra từ 2 đến căn bậc hai của n`
`  {`
`  for (int i = 2; i <= sqrt(n); i++)`
`  {`
`  if (n % i == 0)`
`  {`
`  nguyento = false;`
`  break; // Thoát ngay khi tìm thấy ước`
`  }`
`  }`
`  }`
`  if (nguyento)`
`  {`
`  cout << n << " la so nguyen to" << endl;`
`  }`
`  else`
`  {`
`  cout << n << " KHONG la so nguyen to" << endl;`
`  }`
`  return 0;`
`}`
	- Test 1:
	`Nhap N: 29`
	`29 la so nguyen to`
	- Test 2:
	`Nhap N: 10`
	`10 KHONG la so nguyen to`

2. Bài 2. Sinh dãy Fibonacci.

- **Mô tả:** Nhập số nguyên dương N. In ra N số Fibonacci đầu tiên.
- **Quy tắc:** $F_1 = 1, F_2 = 1, \ldots, F_n = F_{n-1} + F_{n-2}$. (Sử dụng vòng lặp for hoặc while)

2.1. Code Python (Dùng for):

```python
n = int(input("Nhap N so Fibonacci: "))
a, b = 1, 1 # Khởi tạo 2 số đầu tiên
if n >= 1:
    print(a, end=" ")
if n >= 2:
    print(b, end=" ")
for _ in range(2, n):
    next_fibo = a + b
    print(next_fibo, end=" ")
    # Cập nhật a, b cho lần lặp tiếp theo
    a = b
    b = next_fibo
print()
```

=> Kết quả sau khi chạy chương trình:

```
Nhap N so Fibonacci: 6
1 1 2 3 5 8
```

2.2. Code C++ (Dùng for):

```cpp
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
using namespace std;
int main()
{
    int n;
    cout << "Nhap N so Fibonacci: ";
    cin >> n;
    // Khởi tạo 2 số đầu tiên
    long long a = 1, b = 1; // Dùng long long vì dãy Fibo tăng nhanh
    if (n >= 1)
        cout << a << " ";
    if (n >= 2)
        cout << b << " ";
    for (int i = 2; i < n; i++)
    {
        long long next_fibo = a + b;
        cout << next_fibo << " ";
        // Cập nhật a, b cho lần lặp tiếp theo
        a = b;
        b = next_fibo;
    }
    cout << endl;
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
Nhap N so Fibonacci: 6
1 1 2 3 5 8
```

VI. Tổng Kết & Dặn Dò:

1. Củng cố chính:

- **for** dùng khi **biết số lần lặp** (hoặc tập hợp cần duyệt).
- **while** dùng khi lặp dựa trên **điều kiện dừng** (lặp không xác định).
- Trong tính toán số thực, luôn đảm bảo các biến liên quan đến phép chia là **float/double** (ví dụ 1.0 / i).

2. Dặn dò:

Hoàn thành bài tập về nhà và chuẩn bị trước kiến thức về **Mảng (Array)** cho buổi tiếp theo.

CHỦ ĐỀ 2. FILE I/O & KỸ THUẬT LẬP TRÌNH CƠ BẢN

BÀI 4. ĐỌC/GHI FILE CHUẨN (*.INP, *.OUT)

I. Mục Tiêu:

- **Kiến thức:** Học sinh (**HS**) nắm vững cơ chế hoạt động của File I/O trong bối cảnh thi đấu (đọc từ *.INP và ghi ra *.OUT).
- **Kỹ năng:** HS thành thạo các thao tác:
  + Mở/Đóng file an toàn.
  + Đọc nhiều giá trị, đọc mảng từ file.
  + Xử lý dữ liệu và ghi kết quả chính xác ra file output.
- **Tư duy:** Rèn luyện thói quen kiểm tra lỗi I/O và xử lý dữ liệu kiểu chuỗi đọc được thành kiểu số.

II. Phân Bổ Thời Gian (90 Phút):

Hoạt Động	Nội Dung	Thời gian (Phút)
Khởi động	Ôn tập Vòng lặp & Lý thuyết File I/O cơ bản	10
Lý thuyết	Kỹ thuật đọc/ghi file chuẩn (*.INP, *.OUT)	10
Thực hành 1	Bài 1. Đọc mảng, ghi số lớn nhất	25
Thực hành 2	Bài 2. Ghi các số chia hết cho 3	25
Tổng kết	Bài tập về nhà, Củng cố	20
Tổng cộng		**90**

III. Lý Thuyết: Kỹ Thuật Đọc/Ghi File Chuẩn (20 phút)

1. Khái quát chung:

- **Quy ước:** Dữ liệu đầu vào luôn nằm trong file có đuôi **.INP** (ví dụ: **input.INP** hoặc **BAI1.INP**), và kết quả luôn được ghi ra file có đuôi **.OUT** (ví dụ: **output.OUT** hoặc **BAI1.OUT**).
- **Vị trí:** Các file **.INP** và **.OUT** phải nằm **cùng thư mục** với file **code.py** hoặc **code.cpp**.

2. Kỹ thuật Đọc/Ghi Dữ Liệu:

2.1. Python:

2.1.1. Mở/Đóng file:

- Mở file:
  + Dùng **open()**
  + Ví dụ mở file **BAI1.INP**:
    fi = open('BAI1.INP', 'r') => Nhớ là phải có **'r'**
  + Ví dụ mở file **BAI1.OUT**:
    fo = open('BAI1.OUT', 'w') => Bắt buộc phải có 'w'
- Đóng file:
  + Dùng **.close()**
  + Ví dụ đóng file **BAI1.INP**:
    fi.close()

2.1.2. Đọc dữ liệu:

- Đọc cả file, tách chuỗi, ép kiểu.
- Ví dụ đọc/ép kiểu: Nhập mảng **A** có **n** phần tử từ file **.INP**

```
data = fi.read().split()
A = list(map(int, data))
```

2.1.3. Ghi dữ liệu:

- Phải ép kiểu số thành **chuỗi (str)** trước khi ghi.
- Ví dụ:

```
fo.write(f'{ketqua}')
```

2.2. C++:

2.2.1. Mở/Đóng file:

- Mở file:
  + Dùng **ifstream** và **ofstream**
  + Ví dụ mở file **BAI1.INP**:
    ifstream fi("BAI1.INP");
  + Ví dụ mở file **BAI1.OUT**:
    ofstream fo("BAI1.OUT");
- Đóng file:
  + Dùng **.close()**
  + Ví dụ đóng file **BAI1.INP**:
    fi.close()

2.2.2. Đọc dữ liệu:

- Dùng toán tử >> (tự động xử lý khoảng trắng/xuống dòng).
- Ví dụ: Nhập mảng **A** có **n** phần tử từ file **.INP**

```
for (int i = 0; i < n; i++)
{
    fi >> a[i];
}
```

2.2.3. Ghi dữ liệu:

- Tương tự **cout**, ghi trực tiếp giá trị số.
- Ví dụ:

```
fo << ketqua;
```

IV. Thực Hành Kèm Code Tham Khảo:

- Chuẩn bị File Input (Dùng chung cho cả Bài 1 và Bài 2)

- Trước khi thực hành, yêu cầu HS tạo file **INPUT.INP** có nội dung sau:

```
5
10 3 21 15 4
```

+ Dòng 1: Số lượng phần tử N = 5.

+ Dòng 2: Mảng **a** gồm N phần tử, mỗi phần tử trong mảng **a** phải cách nhau ít nhất 1 dấu cách.

1. Bài 1. Đọc mảng từ file, ghi số lớn nhất. (25 phút)

- **Yêu cầu:** Đọc N (số lượng phần tử) và một mảng **a** từ file **INPUT.INP**. Tìm và ghi số lớn nhất của mảng **a** vào file **OUTPUT1.OUT**.

- **Ví dụ:**

INPUT.INP	OUTPUT1.OUT
5 10 3 21 15 4	21

1.1. Code Python:

```python
# 1. Đọc dữ liệu từ file INPUT.INP
# Mở file INPUT.INP để đọc dữ liệu
fi = open('INPUT.INP', 'r') # Nhớ là phải có 'r'
# Đọc toàn bộ các phần tử có trong file, tách thành các phần tử chuỗi
data = fi.read().split()
# Đóng file sau khi đã đọc xong
fi.close()
# 2. Lấy N và mảng a
#   N chính là phần tử đầu tiên của mảng data (data[0])
#   Trong 1 mảng, thì theo quy định phần tử đầu tiên luôn ở vị trí là 0
N = int(data[0])                # Lấy giá trị N
a = list(map(int, data[1:]))    # Mảng A bắt đầu từ data[1] đến hết
# 3. Tìm số lớn nhất
max_val = max(a)
# 4. Ghi kết quả ra file OUTPUT1.OUT
# Mở file OUTPUT1.INP để ghi kết quả
fo = open('OUTPUT1.OUT', 'w')    # Bắt buộc phải có 'w'
# Ghi kết quả tìm được ra file OUTPUT1.OUT
fo.write(f'{max_val}')
# Đóng file sau khi đã ghi kết quả xong
fo.close()
```

- Giải thích Code Python:

Kỹ thuật đọc mảng: Dùng **list(map(int, data[1:]))** để ép kiểu hàng loạt các phần tử từ vị trí **1** trở đi thành số nguyên.

1.2. Code C++:

```cpp
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
#include <fstream>           // Thư viện đọc, ghi file
#include <vector>            // Thư viện dùng cho xử lí mảng động (vector)
using namespace std;
int main()
{
    // 1. Đọc dữ liệu từ file INPUT.INP
    // Mở file INPUT.INP để đọc dữ liệu
    ifstream fi("INPUT.INP");
    // Đọc số lượng phần tử N từ file
    int N;
    fi >> N;
    // Đọc từng phần tử vào vector mảng động a
    vector<int> a(N);
    for (int i = 0; i < N; i++)
    {
        fi >> a[i];
    }
    // Đóng file sau khi đã đọc xong
    fi.close();
    // 2. Tìm số lớn nhất (Dùng hàm có sẵn hoặc tự viết for)
    int max_val = *max_element(a.begin(), a.end());
    // 3. Ghi kết quả ra file OUTPUT1.OUT
    // Mở file OUTPUT1.INP để ghi kết quả
    ofstream fo("OUTPUT1.OUT");
    // Ghi kết quả tìm được ra file OUTPUT1.OUT
    fo << max_val;
    // Đóng file sau khi đã ghi kết quả xong
    fo.close();
    return 0;
}
```

- Giải thích Code C++:

Kỹ thuật đọc mảng: Cần dùng **vector<int> (mảng động)** và vòng **for** để đọc **N** lần. Chú ý **fi >> a[i]** sẽ tự động lấy số tiếp theo từ file.

2. Bài 2. Ghi các số chia hết cho 3. (25 phút)

- Yêu cầu: Tiếp tục sử dụng dữ liệu từ **INPUT.INP**. Tìm tất cả các số trong mảng a chia hết cho 3 và ghi chúng, **mỗi số trên một dòng**, vào file **OUTPUT2.OUT**.

- Ví dụ:

INPUT.INP	OUTPUT2.OUT
5 10 3 21 15 4	3 21 15

2.1. Code Python:

```python
# 1. Đọc dữ liệu từ file INPUT.INP
# Mở file INPUT.INP để đọc dữ liệu
fi = open('INPUT.INP', 'r') # Nhớ là phải có 'r'
# Đọc toàn bộ các phần tử có trong file, tách thành các phần tử chuỗi
data = fi.read().split()
# Đóng file sau khi đã đọc xong
fi.close()
# 2. Lấy N và mảng a
#   N chính là phần tử đầu tiên của mảng data (data[0])
#   Trong 1 mảng, thì theo quy định phần tử đầu tiên luôn ở vị trí là 0
N = int(data[0])                # Lấy giá trị N
a = list(map(int, data[1:]))    # Mảng A bắt đầu từ data[1] đến hết
# 3. Tìm các số chia hết cho 3
ketqua = []
for so in a:
    if so % 3 == 0:
        ketqua.append(str(so))
# 4. Ghi kết quả ra file OUTPUT2.OUT
# Mở file OUTPUT2.INP để ghi kết quả
fo = open('OUTPUT2.OUT', 'w')    # Bắt buộc phải có 'w'
# Ghi kết quả tìm được ra file OUTPUT2.OUT
# Dùng join để nối các phần tử với ký tự xuống dòng
fo.write('\n'.join(ketqua))
# Đóng file sau khi đã ghi kết quả xong
fo.close()
```

- Giải thích Code Python:

Ghi nhiều dòng: Thu thập các kết quả vào một danh sách các chuỗi (**ketqua**), sau đó dùng phương thức **'\n'.join(ketqua)** để ghi tất cả cùng lúc, mỗi phần tử cách nhau bởi dấu xuống dòng.

2.2. Code C++:

```cpp
#include <iostream>         // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>    // Thư viện tổng hợp
#include <fstream>          // Thư viện đọc, ghi file
#include <vector>           // Thư viện dùng cho xử lí mảng động (vector)
using namespace std;
int main()
{
    // 1. Đọc dữ liệu từ file INPUT.INP
    // Mở file INPUT.INP để đọc dữ liệu
    ifstream fi("INPUT.INP");
    // Đọc số lượng phần tử N từ file
    int N;
    fi >> N;
    // Đọc từng phần tử vào vector mảng động a
    vector<int> a(N);
    for (int i = 0; i < N; i++)
    {
        fi >> a[i];
    }
    // Đóng file sau khi đã đọc xong
    fi.close();
    // 2. Mở file OUTPUT2.INP để ghi kết quả
    ofstream fo("OUTPUT2.OUT");
    // Duyệt từ đầu mảng a đến cuối mảng a, tìm các số chia hết cho 3 và ghi ra luôn
    for (int i = 0; i < N; i++)
    {
        if (a[i] % 3 == 0)
        {
            fo << a[i] << endl;
        }
    }
    // Đóng file sau khi đã ghi kết quả xong
    fo.close();
    return 0;
}
```

- Giải thích Code C++:

Ghi nhiều dòng: Sử dụng **fo << a[i] << endl;** trong vòng lặp **if** để in ra số và tự động xuống dòng sau mỗi số thỏa mãn điều kiện.

V. Bài Tập Về Nhà Kèm Lời Giải Tham Khảo:

- Bài tập về nhà. Tách số chẵn/lẻ vào 2 file riêng.

- Yêu cầu: Đọc một mảng số nguyên từ file **DATA.INP**. Ghi tất cả số chẵn vào **CHAN.OUT** và tất cả số lẻ vào file **LE.OUT**. Mỗi số cách nhau bằng dấu cách (trên cùng một dòng).

- Yêu cầu File Input: Tạo file **DATA.INP** (Dòng 1 là N, dòng 2 là N số).

- Ví dụ:

DATA.INP	CHAN.OUT	LE.OUT
6 1 2 3 4 5 6	2 4 6	1 3 5

1. Code Python:

```python
# 1. Đọc dữ liệu từ file DATA.INP
# Mở file DATA.INP để đọc dữ liệu
fi = open('DATA.INP', 'r') # Nhớ là phải có 'r'
# Đọc toàn bộ các phần tử có trong file, tách thành các phần tử chuỗi
```

```
data = fi.read().split()
# Đóng file sau khi đã đọc xong
fi.close()
# 2. Lấy N và mảng a
#   N chính là phần tử đầu tiên của mảng data (data[0])
#   Trong 1 mảng, thì theo quy định phần tử đầu tiên luôn ở vị trí là 0
N = int(data[0])                # Lấy giá trị N
a = list(map(int, data[1:]))    # Mảng A bắt đầu từ data[1] đến hết
# 3. Tìm các số chẵn
chan = []   # Khởi tạo mảng chan để lưu trữ các số chẵn
for i in range(0, N):
    if a[i] % 2 == 0:
        chan.append(a[i])
# 4. Tìm các số lẽ
le = []     # Khởi tạo mảng le để lưu trữ các số lẽ
for i in range(0, N):
    if a[i] % 2 == 1:
        le.append(a[i])
# 5. Ghi kết quả ra file CHAN.OUT
# Mở file CHAN.OUT để ghi kết quả
fo1 = open('CHAN.OUT', 'w')    # Bắt buộc phải có 'w'
# Ghi kết quả tìm được ra file CHAN.OUT
# Duyệt từ đầu đến cuối mảng chan để in ra kết quả, mỗi phần tử cách nhau 1 dấu
cách
# len(chan): Đây là độ dài của mảng chan
for i in range(0, len(chan)):
    fo1.write(f'{chan[i]} ')
# Đóng file fo1 sau khi đã ghi kết quả xong
fo1.close()
# 6. Ghi kết quả ra file LE.OUT
# Mở file LE.OUT để ghi kết quả
fo2 = open('LE.OUT', 'w')    # Bắt buộc phải có 'w'
# Ghi kết quả tìm được ra file LE.OUT
# Duyệt từ đầu đến cuối mảng lẽ để in ra kết quả, mỗi phần tử cách nhau 1 dấu cách
# len(le): Đây là độ dài của mảng le
for i in range(0, len(le)):
    fo2.write(f'{le[i]} ')
# Đóng file fo2 sau khi đã ghi kết quả xong
fo2.close()
```

2. Code C++:

```
#include <iostream>         // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>    // Thư viện tổng hợp
#include <fstream>          // Thư viện đọc, ghi file
#include <vector>           // Thư viện dùng cho xử lí mảng động (vector)
using namespace std;
int main()
{
    // 1. Đọc dữ liệu từ file DATA.INP
    // Mở file DATA.INP để đọc dữ liệu
    ifstream fi("DATA.INP");
    // Đọc số lượng phần tử N từ file
    int N;
    fi >> N;
    // Đọc từng phần tử vào vector mảng động a
    vector<int> a(N);
    for (int i = 0; i < N; i++)
```

```
    {
        fi >> a[i];
    }
    // Đóng file sau khi đã đọc xong
    fi.close();
    // 2. Mở file CHAN.OUT để ghi kết quả
    ofstream fo1("CHAN.OUT");
    // Duyệt từ đầu mảng a đến cuối mảng a, tìm các số chẵn và ghi ra luôn
    for (int i = 0; i < N; i++)
    {
        if (a[i] % 2 == 0)
        {
            fo1 << a[i] << " ";
        }
    }
    // Đóng file sau khi đã ghi kết quả xong
    fo1.close();
    // 3. Mở file LE.OUT để ghi kết quả
    ofstream fo2("LE.OUT");
    // Duyệt từ đầu mảng a đến cuối mảng a, tìm các số lẻ và ghi ra luôn
    for (int i = 0; i < N; i++)
    {
        if (a[i] % 2 == 1)
        {
            fo2 << a[i] << " ";
        }
    }
    // Đóng file sau khi đã ghi kết quả xong
    fo2.close();
    return 0;
}
```

VI. Tổng Kết & Dặn Dò:

1. Củng cố chính:

- **Luôn kiểm tra** đã đọc hết dữ liệu **N** phần tử chưa.
- **Python:** Nhớ dùng **split()** để đọc dễ nhất và **close()** file sau khi dùng.
- **C++:** Luôn **include <fstream>** và **close()** file sau khi dùng.

2. Dặn dò: Hoàn thành bài tập về nhà và chuẩn bị trước kiến thức về **Mảng (Array)** và **Hàm (Function)** cho buổi tiếp theo, vì chúng ta đã bắt đầu làm việc với các tập hợp dữ liệu lớn.

BÀI 5. HÀM VÀ TÁI SỬ DỤNG MÃ

I. Mục Tiêu:

- **Kiến thức:** Học sinh (**HS**) nắm vững khái niệm về **Hàm** (Function) và tầm quan trọng của việc **tái sử dụng mã** (Code Reusability).
- **Kỹ năng:** HS thành thạo cách:
  + **Định nghĩa** (Define) và **Gọi** (Call) hàm có và không có tham số (parameters).
  + Hiểu và sử dụng lệnh **return** để trả về giá trị.
  + Phân biệt vai trò của các hàm trong Python và C++.

- **Tư duy:** Rèn luyện thói quen chia nhỏ bài toán phức tạp thành các hàm nhỏ, độc lập.

II. Phân Bổ Thời Gian (90 Phút):

Hoạt Động	Nội Dung	Thời gian (Phút)
Khởi động	Ôn tập Vòng lặp & I/O File	10
Lý thuyết 1	Khái niệm Hàm, Lợi ích, Cú pháp cơ bản	15
Lý thuyết 2	Tham số, Đối số, Lệnh **return**	15
Thực hành 1	Bài 1. Hàm kiểm tra số chẵn/lẻ	20
Thực hành 2	Bài 2. Tái sử dụng hàm (Tính diện tích)	20
Tổng kết	Bài tập về nhà, Củng cố	10
Tổng cộng		**90**

III. Nội Dung Giảng Dạy Chi Tiết & Demo:

1. Khái Niệm Hàm & Cú Pháp Cơ Bản (15 phút):

1.1. Hàm:

- Khối lệnh được đặt tên, thực hiện một nhiệm vụ cụ thể. Giúp mã gọn gàng và dễ bảo trì.
- Python: Định nghĩa bằng từ khóa **def**.
- C++: Định nghĩa bằng **Kiểu Trả Về + Tên Hàm**.

1.2. Hàm không tham số, không trả về:

- Hàm chỉ thực hiện in ấn hoặc tác vụ đơn giản.
- Ví dụ Python:

```python
def chao_mung():
    print("Hello!")
# Gọi hàm
chao_mung()
```

=> Kết quả in ra sau khi chạy chương trình:

```
Hello!
```

- Ví dụ C++:

```cpp
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
using namespace std;
void chao_mung()
{
    cout << "Hello!" << endl;
}
int main()
{
    // Gọi hàm
    chao_mung();
    return 0;
}
```

=> Kết quả in ra sau khi chạy chương trình:

```
Hello!
```

1.3. Lưu ý C++: Hàm main **bắt buộc** phải là **int main()** và phải nằm trong **main** để chương trình nhận diện.

2. Tham Số, Đối Số và return (15 phút):

- **Tham số (Parameters):** Các biến được định nghĩa trong dấu ngoặc đơn (...) của hàm, dùng để nhận dữ liệu.
- **Đối số (Arguments):** Giá trị thực tế được truyền vào khi hàm được gọi.
- **return:** Lệnh thoát khỏi hàm và **trả về một giá trị** cho nơi gọi hàm.

2.1. Hàm có return:

2.1.1. Python:

- Dùng từ khóa **return**. Nếu không có **return**, hàm trả về **None**.
- Ví dụ Python:

```python
def tinh_tong(a, b):
    return a + b
ket_qua = tinh_tong(5, 3)
print(ket_qua)  # ket_qua = 8
```

2.1.2. C++:

- Phải khai báo kiểu trả về (ví dụ **int**) thay vì **void**.
- Ví dụ C++:

```cpp
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
using namespace std;
int tinh_tong(int a, int b)
{
    return a + b;
}
int main()
{
    int ket_qua = tinh_tong(5, 3);
    cout << ket_qua;     // ket_qua = 8
    return 0;
}
```

2.2. Hàm không trả về:

2.2.1. Python:

- Hàm không có **return** hoặc **return** không giá trị.
- Ví dụ Python:

```python
def in_ra_a(a):
    print(a)
```

2.2.2. C++:

- Phải dùng từ khóa **void** (nghĩa là "không có giá trị").
- Ví dụ C++:

```cpp
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
using namespace std;
void in_ra_a(int a)
{
    cout << a << endl;
}
int main()
{

    return 0;
}
```

2.3. Nguyên mẫu (Prototype):

2.3.1. Python:

Không cần nguyên mẫu, chỉ cần định nghĩa hàm **trước** khi gọi.

2.3.2. C++:

- Bắt buộc phải khai báo nguyên mẫu (hoặc định nghĩa hàm) **trước** hàm **main**.
- Ví dụ C++:

```cpp
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
using namespace std;
int tinh_tong(int a, int b)
{
    return a + b;
}
int main()
{
    int ket_qua = tinh_tong(5, 3);
    cout << ket_qua;     // ket_qua = 8
    return 0;
}
```

IV. Thực Hành Kèm Lời Giải Tham Khảo:

1. Bài 1. Hàm kiểm tra số chẵn/lẻ. (20 phút)

- **Yêu cầu:** Viết hàm **is_even(num)** nhận vào một số nguyên **num**, trả về **True/False** (hoặc **1/0**) cho biết số đó có phải là số chẵn hay không.

1.1. Code Python:

```python
def is_even(num):   # Hàm trả về True nếu chia hết cho 2, False nếu không
    return num % 2 == 0
# Kiểm tra
print(f"10 chan le: {is_even(10)}") # True
print(f"11 chan le: {is_even(11)}") # False
```

=> Kết quả in ra sau khi chạy chương trình:

```
10 chan le: True
11 chan le: False
```

- **Giải thích Code Python:** Hàm trả về giá trị **Boolean** (**True/False**) trực tiếp từ biểu thức so sánh.

1.2. Code C++:

```cpp
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
using namespace std;
// Hàm trả về kiểu bool (true/false) hoặc int (1/0)
bool is_even(int num)
{
    return num % 2 == 0;
}
int main()
{
    if (is_even(10))
    {
        cout << "10 la so chan" << endl;
    }
    else
    {
        cout << "10 la so le" << endl;
    }
    return 0;
}
```

=> Kết quả in ra sau khi chạy chương trình:

```
10 la so chan
```

- Giải thích Code C++:

+ Hàm trả về kiểu **bool**.

+ Nếu dùng **int** làm kiểu trả về, sẽ **return 1** cho **True** và **return 0** cho **False**.

2. Bài 2. Tái sử dụng hàm (Tính diện tích). (20 phút)

- **Yêu cầu:** Viết 2 hàm: **tinh_dien_tich_HV(a)** (hình vuông) và **tinh_dien_tich_TG(a, h)** (tam giác). Sau đó, viết hàm **tong_dien_tich** gọi lại 2 hàm trên để tính tổng diện tích.

2.1. Code Python:

```python
# Ham 1: Tinh DT Hinh Vuong
def tinh_dien_tich_HV(a):
    return a * a
# Ham 2: Tinh DT Tam Giac
def tinh_dien_tich_TG(a, h):
    return 0.5 * a * h
# Ham 3: Goi lai 2 ham tren (Tai su dung)
def tong_dien_tich(canh_hv, day_tg, chieu_cao_tg):
    dt_hv = tinh_dien_tich_HV(canh_hv)
    dt_tg = tinh_dien_tich_TG(day_tg, chieu_cao_tg)
    return dt_hv + dt_tg
# Goi ham chinh
ket_qua = tong_dien_tich(10, 5, 8)
print(f"Tong dien tich: {ket_qua}")     # 100 + 20 = 120
```

=> Kết quả in ra sau khi chạy chương trình:

```
Tong dien tich: 120.0
```

- Giải thích Code Python:

Tái sử dụng: Hàm **tong_dien_tich** không tự tính toán mà **gọi lại** các hàm nhỏ hơn, thể hiện tính **module hóa** và **tái sử dụng mã**.

2.2. Code C++:

```
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
using namespace std;
// 1. ĐỊNH NGHĨA CÁC HÀM PHỤ TRƯỚC HÀM MAIN
double tinh_dien_tich_HV(double a)
{
    return a * a;
}
double tinh_dien_tich_TG(double a, double h)
{
    return 0.5 * a * h;
}
double tong_dien_tich(double a, double day, double h)
{
    // Các hàm phụ đã được định nghĩa ở trên, nên việc gọi là hợp lệ
    double dt_hv = tinh_dien_tich_HV(a);
    double dt_tg = tinh_dien_tich_TG(day, h);
    return dt_hv + dt_tg;
}
// 2. HÀM CHÍNH (MAIN FUNCTION)
int main()
{
    // Hàm main gọi hàm tong_dien_tich
    double ket_qua = tong_dien_tich(10, 5, 8);
    cout << "Tong dien tich: " << ket_qua;
    return 0;
}
```

=> Kết quả in ra sau khi chạy chương trình:

```
Tong dien tich: 120
```

- Giải thích Code C++: Trong **C++**, **phải khai báo các hàm con trước hàm main** để trình biên dịch biết hàm đó tồn tại.

V. Bài Tập Về Nhà Kèm Gợi Ý Code:

1. Bài 1. Hàm kiểm tra số nguyên tố.

- Mô tả: Chuyển đổi logic kiểm tra số nguyên tố (đã làm ở Buổi 3) thành hàm **nguyento(N)**, trả về **True/False**.

1.1. Code Python:

```python
def nguyento(N):
    if N < 2: return False
    for i in range(2, int(N**0.5) + 1):
        if N % i == 0:
            return False # Thoát ngay khi tìm thấy ước
    return True
# Kiểm tra
print(f"7 la SNT: {nguyento(7)}")
```

=> Kết quả in ra sau khi chạy chương trình:

```
7 la SNT: True
```

1.2. Code C++:

```cpp
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
using namespace std;
bool nguyento(int N)
{
    if (N < 2) return false;
    for (int i = 2; i <= sqrt(N); i++)
    {
        if (N % i == 0)
        {
            return false;
        }
    }
    return true;
}
int main()
{
    if (nguyento(7))
        cout << "7 la SNT";
    return 0;
}
```

=> Kết quả in ra sau khi chạy chương trình:

```
7 la SNT
```

2. Bài 2. Hàm tính tổng dãy Fibonacci.

- **Mô tả:** Viết hàm **sum_fibo(N)** nhận số nguyên **N**, tính và trả về **tổng** của **N** số Fibonacci đầu tiên.

2.1. Code Python:

```python
def sum_fibo(N):
    # 1. Điều kiện cơ sở:
    if N <= 0:
        return 0
    if N == 1:
        return 1    # Tổng 1 số đầu tiên (F1) là 1
    # 2. Khởi tạo: F1 = 1, F2 = 1
    a, b = 1, 1
    current_sum = 2 # Tổng ban đầu của F1 + F2 = 1 + 1 = 2
    # 3. Vòng lặp tính từ F3 den Fn (N-2 lan lap)
    # Range chạy từ 2 đến N-1 (tổng cộng N-2 lần)
    for _ in range(2, N):
        next_fibo = a + b                       # Tính Fi (bắt đầu từ F3)
        current_sum = current_sum + next_fibo   # Cộng Fi vào tổng
        # Cập nhật F(i-2) = F(i-1) và F(i-1) = Fi
        a, b = b, next_fibo
    return current_sum  # Trả về tổng N số Fibo
# Kiểm tra: Tổng 5 số Fibo đầu tiên (F1, F2, F3, F4, F5) = 1 + 1 + 2 + 3 + 5 = 12
print(f"Tong 5 so Fibo dau tien: {sum_fibo(5)}")
```

=> Kết quả in ra sau khi chạy chương trình:

```
Tong 5 so Fibo dau tien: 12
```

2.2. Code C++:

```cpp
#include <iostream>         // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>    // Thư viện tổng hợp
using namespace std;
long long sum_fibo(int N)
{
    // 1. Điều kiện cơ sở:
    if (N <= 0) return 0;
    if (N == 1) return 1;   // Tổng 1 số đầu tiên (F1) là 1
    // 2. Khởi tạo: F1 = 1, F2 = 1
    long long a = 1, b = 1;
    long long current_sum = 2;  // Tổng ban đầu của F1 + F2 = 1 + 1 = 2
    // 3. Vòng lặp tính từ F3 den Fn (N-2 lan lap)
    for (int i = 3; i <= N; i++)
    {
        long long next_fibo = a + b;            // Tính Fi (bắt đầu từ F3)
        current_sum = current_sum + next_fibo;  // Cộng Fi vào tổng
        // Cập nhật F(i-2) = F(i-1) và F(i-1) = Fi
        a = b;
        b = next_fibo;
    }
    return current_sum; // Trả về tổng N số Fibo
}
int main()
{
    // Kiểm tra: Tổng 5 số Fibo đầu tiên (F1, F2, F3, F4, F5) = 1 + 1 + 2 + 3 + 5 = 12
    cout << "Tong 5 so Fibo dau tien: " << sum_fibo(5) << endl;
    return 0;
}
```

=> Kết quả in ra sau khi chạy chương trình:

```
Tong 5 so Fibo dau tien: 12
```

VI. Tổng Kết & Dặn Dò:

1. Củng cố chính:

- **Hàm** giúp tái sử dụng mã và quản lý chương trình lớn.
- Luôn xác định: **Hàm cần gì?** (Tham số) và **Hàm trả về gì?** (**return** và kiểu trả về).
- **C++** bắt buộc cần kiểu trả về (hoặc void) và nguyên mẫu (hoặc định nghĩa trước main).

2. Dặn dò: Hoàn thành bài tập và chuẩn bị cho buổi học tiếp theo về Thuật toán cơ bản: **min/max, đếm, tổng dãy**.

BÀI 6. THUẬT TOÁN CƠ BẢN: MIN/MAX, ĐẾM, TỔNG DÃY

I. Mục Tiêu:

- **Kiến thức:** Học sinh (**HS**) hiểu và nắm vững nguyên lý hoạt động của các thuật toán: **Tìm giá trị nhỏ nhất/lớn nhất** (min/max), **Tính tổng dãy**, và **Đếm phần tử** theo điều kiện.
- **Kỹ năng:** HS thành thạo cách sử dụng vòng lặp (for) để **duyệt toàn bộ mảng/danh sách** và cập nhật kết quả.
- **Tư duy:** Rèn luyện thói quen **khởi tạo giá trị ban đầu** (cho min/max hoặc tổng) một cách hợp lý trước khi bắt đầu vòng lặp.

II. Phân Bổ Thời Gian (90 Phút):

Hoạt Động	Nội Dung	Thời gian (Phút)
Khởi động	Ôn tập Hàm & Giới thiệu về duyệt mảng	10
Lý thuyết 1	Nguyên tắc tìm min/max và Tính tổng	15
Thực hành 1	Bài 1. Tìm min/max và vị trí	25
Thực hành 2	Bài 2. Đếm số dương và Tính tổng các số âm	25
Tổng kết	Bài tập về nhà, Củng cố	15
Tổng cộng		**90**

III. Lý Thuyết: Nguyên Tắc Duyệt Dãy/Mảng (25 phút)

1. Khởi tạo và Duyệt Mảng:

- **Python:** Dùng **Danh sách (List)**. Duyệt dễ dàng bằng **for item in list** hoặc **for i in range(len(list))**.
- **C++:** Dùng **Mảng** (kích thước cố định) hoặc **Vector** (mảng động). Duyệt bằng **for (init; condition; step)**.
- **Tìm Max**:
  + Nguyên tắc khởi tạo (Init):
    Gán **max = phần tử đầu tiên của dãy**.
  + Giải thích: Không nên gán **max = 0** vì nếu tất cả các số đều âm, kết quả sẽ sai.
- **Tìm Min:**
  + Nguyên tắc khởi tạo (Init):
    Gán **min = phần tử đầu tiên của dãy**.
  + Giải thích: Không nên gán **min** bằng một số lớn vô lý.

- **Tính Tổng:**
  + Nguyên tắc khởi tạo (Init):
    Gán **sum = 0**.
  + Giải thích: Bắt đầu tích lũy giá trị từ 0.
- **Đếm:**
  + Nguyên tắc khởi tạo (Init):
    Gán **count = 0**.
  + Giải thích: Bắt đầu đếm từ 0.

2. Code Tham Khảo Khởi tạo và Duyệt (Giả sử mảng a)

2.1. Python:

```python
a = [10, 5, 20]
# Khởi tạo
max_val = a[0]
# Duyệt
for num in a:
    if num > max_val:
        max_val = num
# In ket qua
print(max_val)  # Kết quả sẽ là: max_val = 20
```

2.2. C++:

```cpp
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
using namespace std;
int main()
{
    // Khai báo Vector
    vector<int> a = {10, 5, 20};
    // Khởi tạo
    int max_val = a[0];
    // Duyệt bằng chỉ số
    for (int i = 1; i < a.size(); i++)
    {
        if (a[i] > max_val)
        {
            max_val = a[i];
        }
    }
    // In kết quả
    cout << max_val;    // Kết quả: max_val = 20
    return 0;
}
```

IV. Thực Hành Kèm Lời Giải Tham Khảo:

1. Bài 1. Tìm min/max và vị trí. (25 phút)

- **Yêu cầu:** Cho mảng **A**. Tìm giá trị **lớn nhất** và **vị trí** của nó trong mảng.
- **Mảng Input:** A = [5, 12, 3, 12, 8] (Vị trí max là 1 và 3).

1.1. Code Python:

```python
A = [5, 12, 3, 12, 8]
max_val = A[0]
vi_tri = []    # Khởi tạo list ghi lại vị trí max_val
# Duyệt mảng
for i in range(0, len(A)):
    if A[i] > max_val:
        max_val = A[i]      # Cập nhật MAX mới
        vi_tri = [i]        # Reset vị trí mới
    elif A[i] == max_val:
        vi_tri.append(i)    # Thêm vị trí nếu trùng MAX
print(f"Max: {max_val}")
print(f"Vi tri: {vi_tri}")
```

=> Kết quả sau khi chạy chương trình:

```
Max: 12
Vi tri: [1, 3]
```

- **Giải thích Code Python:** Khởi tạo **max_val = A[0]**. Dùng một **list** để lưu các vị trí có cùng giá trị **max**.

1.2. Code C++:

```cpp
#include <iostream>         // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>    // Thư viện tổng hợp
using namespace std;
int main()
{
    vector<int> A = {5, 12, 3, 12, 8};
    int max_val = A[0];
    vector<int> vi_tri;
    for (int i = 0; i < A.size(); i++)
    {
        if (A[i] > max_val)
        {
            max_val = A[i];
            vi_tri.clear();         // Xóa các vị trí cũ
            vi_tri.push_back(i);    // Thêm vị trí mới
        }
        else if (A[i] == max_val)
        {
            vi_tri.push_back(i);    // Thêm vị trí trùng
        }
    }
    cout << "Max: " << max_val << endl;
    cout << "Vi tri:";
    for (int pos : vi_tri)
    {
        cout << " " << pos;         // Duyet vector vi_tri
    }
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
Max: 12
Vi tri: 1 3
```

- **Giải thích Code C++:** Trong C++, dùng **vector<int>** cho cả mảng **A** và mảng **vi_tri**. Dùng **vi_tri.clear()** để xóa các vị trí cũ khi tìm thấy **max** mới.

2. Bài 2. Đếm số dương và Tính tổng các số âm. (25 phút)

- **Yêu cầu:** Cho mảng B = [10, -5, 2, -8, 15, -1].
  + Đếm số lượng phần tử dương.
  + Tính tổng các phần tử âm.

2.1. Code Python:

```python
B = [10, -5, 2, -8, 15, -1]
count_duong = 0
sum_am = 0
for num in B:
    if num > 0:     # Dieu kien so duong
        count_duong = count_duong + 1
    elif num < 0:   # Dieu kien so am
        sum_am = sum_am + num
print(f"So luong so duong: {count_duong}")  # 3
print(f"Tong so am: {sum_am}")  # -14
```

=> Kết quả sau khi chạy chương trình:

```
So luong so duong: 3
Tong so am: -14
```

- **Giải thích Code Python:** Khởi tạo **count_duong = 0** và **sum_am = 0**. Sử dụng **if/elif** để tách điều kiện.

2.2. Code C++:

```cpp
#include <iostream>         // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>    // Thư viện tổng hợp
using namespace std;
int main()
{
    vector<int> B = {10, -5, 2, -8, 15, -1};
    int count_duong = 0;
    int sum_am = 0;
    // Duyệt bằng for-each (tương tự Python)
    for (int num : B)
    {
        if (num > 0)
        {
            count_duong = count_duong + 1;
        }
        else if (num < 0)
        {
            sum_am = sum_am + num;
        }
    }
    cout << "So luong so duong: " << count_duong << endl; // 3
    cout << "Tong so am: " << sum_am << endl; // -14
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
So luong so duong: 3
Tong so am: -14
```

- **Giải thích Code C++:** C++ có thể dùng cú pháp **for (int num : B)** (for-each loop) rất tiện lợi để duyệt qua từng phần tử của **vector**.

V. Bài Tập Về Nhà Kèm Gợi Ý Code:

1. Bài 1. Tìm phần tử chẵn lớn nhất.

- **Yêu cầu:** Nhập một mảng A. Tìm và in ra số chẵn lớn nhất trong mảng. Nếu không có số chẵn, in ra -1.

1.1. Code Python:

```python
A = [5, 12, 3, 9, 6]
max_chan = -1   # Khởi tạo bằng -1 để xử lý trường hợp không có số chẵn
for num in A:
    if num % 2 == 0:    # Là số chẵn
        if num > max_chan:
            max_chan = num
print(f"So chan lon nhat: {max_chan}")  # 12
```

=> Kết quả sau khi chạy chương trình:

```
So chan lon nhat: 12
```

1.2. Code C++:

```cpp
#include <iostream>         // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>    // Thư viện tổng hợp
using namespace std;
int main()
{
    vector<int> A = {5, 12, 3, 9, 6};
    int max_chan = -1;      // Khởi tạo bằng -1 để xử lý trường hợp không có số chẵn
    for (int num : A)
    {
        if (num % 2 == 0)   // Là số chẵn
        {
            if (num > max_chan)
            {
                max_chan = num;
            }
        }
    }
    cout << "So chan lon nhat: " << max_chan << endl; // 12
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
So chan lon nhat: 12
```

2. Bài 2. Đếm tần suất xuất hiện của một giá trị.

- **Yêu cầu:** Nhập mảng A và một giá trị X. Đếm xem giá trị X xuất hiện bao nhiêu lần trong mảng A.

2.1. Code Python:

```python
A = [1, 5, 2, 5, 3, 5]
X = 5
count = 0
for num in A:
    if num == X:
        count = count + 1
print(f"Tan suat cua {X}: {count}") # 3
```

=> Kết quả sau khi chạy chương trình:

```
Tan suat cua 5: 3
```

2.2. Code C++:

```cpp
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
using namespace std;
int main()
{
    vector<int> A = {1, 5, 2, 5, 3, 5};
    int X = 5;
    int count = 0;
    for (int num : A)
    {
        if (num == X)
        {
            count = count + 1;
        }
    }
    cout << "Tan suat cua " << X << ": " << count << endl; // 3
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
Tan suat cua 5: 3
```

VI. Tổng Kết & Dặn Dò:

1. Củng cố chính:

- **Quan trọng nhất:** Luôn **khởi tạo** biến **min/max/sum/count** trước khi bắt đầu lặp.
- **min/max** thường được khởi tạo bằng phần tử đầu tiên của dãy.
- **Python** hỗ trợ hàm **min(), max(), sum() và count()** có sẵn, nhưng HS cần nắm vững thuật toán bằng vòng lặp.

2. Dặn dò: Hoàn thành bài tập về nhà và chuẩn bị cho buổi học tiếp theo về **UCLN, BCNN**.

CHỦ ĐỀ 3. SỐ HỌC: ƯỚC, BỘI, NGUYÊN TỐ, PHÂN TÍCH THỪA SỐ NGUYÊN TỐ

BÀI 7. ƯỚC CHUNG LỚN NHẤT VÀ BỘI CHUNG NHỎ NHẤT

I. Mục Tiêu:

- **Kiến thức:** Học sinh (**HS**) nắm vững định nghĩa toán học của **UCLN** và **BCNN**.
- **Kỹ năng:**
  + HS thành thạo cách cài đặt thuật toán **Euclid** để tìm **UCLN** (phương pháp hiệu quả nhất).
  + HS biết cách sử dụng mối quan hệ giữa UCLN và BCNN để tính BCNN.
- **Tư duy:** Rèn luyện thói quen sử dụng Hàm (Functions) để module hóa thuật toán.

II. Phân Bổ Thời Gian (90 Phút):

Hoạt Động	Nội Dung	Thời gian (Phút)
Khởi động	Ôn tập Thuật toán min/max	10
Lý thuyết 1	Định nghĩa UCLN, BCNN và Thuật toán Euclid	15
Thực hành 1	Bài 1. Hàm tìm UCLN của hai số	25
Thực hành 2	Bài 2. Hàm tìm BCNN và UCLN của ba số	30
Tổng kết	Bài tập về nhà, Củng cố	10
Tổng cộng		**90**

III. Lý Thuyết: UCLN, BCNN và Thuật toán Euclid (25 phút)

1. Định Nghĩa Cơ Bản:

- **Ước chung lớn nhất (UCLN/GCD):**
  + Định nghĩa: Số nguyên dương lớn nhất là ước của cả hai hay nhiều số. Ký hiệu: **UCLN(a, b)**.
  + Công thức quan trọng: Thuật toán **Euclid**
- **Bội chung nhỏ nhất (BCNN/LCM):**
  + Số nguyên dương nhỏ nhất là bội của cả hai hay nhiều số. Ký hiệu: **BCNN(a, b)**.
  + Công thức: **BCNN(a, b) = a * b / UCLN(a, b)**

2. Thuật toán Euclid (Tìm UCLN):

- Đây là thuật toán tối ưu nhất để tìm **UCLN** của hai số nguyên dương **a** và **b**.
- **Nguyên tắc: UCLN(a, b) = UCLN(b, a mod b)**.
- **Quy trình lặp:** Lặp cho đến khi số dư (**a mod b**) bằng **0**. Khi đó, số **b** (số chia) chính là **UCLN**.

Bước	a	b	a mod b	Kết quả
Khởi tạo	24	18	24 mod 18 = 6	UCLN(24, 18) = UCLN(18, 6)
Lần 1	18	6	18 mod 6 = 0	UCLN(18, 6) = 6
Kết quả				UCLN(**24, 18**) = **6**

IV. Thực Hành Kèm Code Tham Khảo:

1. Bài 1. Hàm tìm UCLN của hai số. (25 phút)

- **Yêu cầu:** Viết hàm **tim_ucln(a, b)** sử dụng thuật toán **Euclid** (dùng vòng lặp **while**).

1.1. Code Python:

```python
def tim_ucln(a, b):
    # Đảm bảo làm việc với giá trị tuyệt đối (UCLN luôn dương)
    a, b = abs(a), abs(b)
    # Thuật toán Euclid: UCLN(a, b) = UCLN(b, a % b)
    while b != 0:
        a, b = b, a % b
    return a
# Kiểm tra
print(f"UCLN(24, 18): {tim_ucln(24, 18)}")
print(f"UCLN(15, 7): {tim_ucln(15, 7)}")
print(f"UCLN(-30, 18): {tim_ucln(-30, 18)}")
```

=> Kết quả sau khi chạy chương trình:

```
UCLN(24, 18): 6
UCLN(15, 7): 1
UCLN(30, -18): 6
```

- **Giải thích Code Python: a, b = b, a % b** là cách thức hiện đại và gọn gàng để hoán đổi giữa **a với b** và **b với a mod b** trong Python. Vòng lặp dừng khi **b = 0**.

1.2. Code C++:

```cpp
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
using namespace std;
long long tim_ucln(long long a, long long b)
{
    // 1. Đảm bảo làm việc với giá trị tuyệt đối (UCLN luôn dương)
    a = abs(a);
    b = abs(b);
    // 2. Thuật toán Euclid (Sử dụng biến tạm 'temp' để hoán đổi giá trị)
    long long temp;
    while (b != 0)
    {
        temp = a % b;    // temp = so du (r)
        a = b;           // a = so chia (b)
        b = temp;        // b = so du moi (r)
    }
    return a;   // Khi b = 0, UCLN là a
}
// Hàm chính để kiểm tra
int main()
{
    cout << "UCLN(24, 18): " << tim_ucln(24, 18) << endl;     // Kết quả: 6
    cout << "UCLN(15, 7): " << tim_ucln(15, 7) << endl;       // Kết quả: 1
    cout << "UCLN(-30, 18): " << tim_ucln(-30, 18) << endl;   // Kết quả: 6
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
UCLN(24, 18): 6
UCLN(15, 7): 1
UCLN(30, -18): 6
```

- **Giải thích Code C++:** C++ cần biến trung gian **temp (r)** để lưu **a mod b** trước khi gán. Sử dụng **abs()** từ thư viện **<cmath>** để đảm bảo **a, b** là số dương.

2. Bài 2. Hàm tìm BCNN và UCLN của ba số. (30 phút)

- **Yêu cầu 1 (BCNN):** Viết hàm **tim_bcnn(a, b)** sử dụng công thức:

  BCNN(a, b) = |a * b| / UCLN(a, b)

- **Yêu cầu 2 (UCLN 3 số):** Viết hàm **tim_ucln_3so(a, b, c)** sử dụng tính chất:

  UCLN(a, b, c) = UCLN(UCLN(a, b), c)

2.1. Code Python:

```python
# Ham tim UCLN
def tim_ucln(a, b):
    # Đảm bảo làm việc với giá trị tuyệt đối (UCLN luôn dương)
    a, b = abs(a), abs(b)
    # Thuật toán Euclid lặp
    while b:
        a, b = b, a % b
    return a
# Ham tim BCNN cua 2 so
def tim_bcnn(a, b):
    if a == 0 or b == 0:
        # BCNN của bất kỳ số nào với 0 luôn là 0
        return 0
    # Công thức: BCNN(a, b) = (|a * b|) / UCLN(a, b)
    # Python dùng // (chia lấy nguyên) để đảm bảo kết quả là số nguyên
    # abs() đảm bảo kết quả luôn dương, theo định nghĩa của BCNN
    return abs(a * b) // tim_ucln(a, b)
# Ham tim UCLN cua 3 so
def tim_ucln_3so(a, b, c):
    a1 = tim_ucln(a, b)
    b1 = c
    return tim_ucln(a1, b1)
# Kiểm tra
print(f"UCLN(12, 18): {tim_ucln(12, 18)}")                  # Kết quả: 6
print(f"BCNN(12, 18): {tim_bcnn(12, 18)}")                  # Kết quả: 36
print(f"BCNN(4, 6): {tim_bcnn(4, 6)}")                      # Kết quả: 12
print(f"BCNN(-10, 5): {tim_bcnn(-10, 5)}")                  # Kết quả: 10
print(f"UCLN(12, 18, 21): {tim_ucln_3so(12, 18, 21)}")      # Kết quả: 3
print(f"UCLN(-10, 5, -30): {tim_ucln_3so(-10, 5, -30)}")    # Kết quả: 5
```

=> Kết quả sau khi chạy chương trình:

```
UCLN(12, 18): 6
BCNN(12, 18): 36
BCNN(4, 6): 12
BCNN(-10, 5): 10
UCLN(12, 18, 21): 3
UCLN(-10, 5, -30): 5
```

2.2. Code C++:

```
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
using namespace std;
// Ham tim UCLN
long long tim_ucln(long long a, long long b)
{
    // Đảm bảo làm việc với giá trị tuyệt đối (UCLN luôn dương)
    a = abs(a);
    b = abs(b);
    // Thuật toán Euclid lặp
    long long temp;
    while (b != 0)
    {
        temp = a % b;    // temp = so du (r)
        a = b;           // a = so chia (b)
        b = temp;        // b = so du moi (r)
    }
    return a;    // Khi b = 0, UCLN là a
}
// Ham tim BCNN cua 2 so
long long tim_bcnn(long long a, long long b)
{
    // BCNN của bất kỳ số nào với 0 luôn là 0
    if (a == 0 || b == 0)
    {
        return 0;
    }
    // Công thức: BCNN = (|a * b|) / UCLN(a, b)
    // Phép nhân a*b có thể gây tràn số, nên dùng long long là bắt buộc.
    // LƯU Ý: Phải tính UCLN(a, b) trước một lần.
    long long ucln_val = tim_ucln(a, b);
    // Vì UCLN(a, b) đã được đảm bảo là dương, ta chỉ cần abs(a * b)
    // Và phép chia / trong C++ khi dùng long long sẽ tự động lấy phần nguyên.
    return abs(a * b) / ucln_val;
}
// Ham tim UCLN cua 3 so
long long tim_ucln_3so(long long a, long long b, long long c)
{
    long long a1 = tim_ucln(a, b);
    long long b1 = c;
    return tim_ucln(a1, b1);
}
// Hàm chính để kiểm tra
int main()
{
    // Cần ép kiểu rõ ràng sang long long nếu nhập số nguyên nhỏ
    cout << "UCLN(12, 18): " << tim_ucln(12, 18) << endl;
    cout << "BCNN(12, 18): " << tim_bcnn(12, 18) << endl;
    cout << "BCNN(4, 6): " << tim_bcnn(4, 6) << endl;
    cout << "BCNN(-10, 5): " << tim_bcnn(-10, 5) << endl;
    cout << "UCLN(12, 18, 21): " << tim_ucln_3so(12, 18, 21) << endl;
    cout << "UCLN(-10, 5, -30): " << tim_ucln_3so(-10, 5, -30) << endl;
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
UCLN(12, 18): 6
BCNN(12, 18): 36
BCNN(4, 6): 12
BCNN(-10, 5): 10
UCLN(12, 18, 21): 3
UCLN(-10, 5, -30): 5
```

V. Bài Tập Về Nhà Kèm Gợi Ý Code:

1. Bài 1. Rút gọn phân số.

- **Mô tả:** Nhập tử số A và mẫu số B. Rút gọn phân số A / B và in ra kết quả.
- **Logic:** Rút gọn bằng cách chia cả tử và mẫu cho UCLN(A, B).

1.1. Code Python:

```python
# Hàm tìm UCLN của 2 số a, b
def tim_ucln(a, b):
    # Đảm bảo làm việc với giá trị tuyệt đối (UCLN luôn dương)
    a, b = abs(a), abs(b)
    # Thuật toán Euclid lặp
    while b:
        a, b = b, a % b
    return a
# Kiểm tra
A = 12
B = 18
UCLN = tim_ucln(A, B)
tu_moi = A // UCLN      # Chia lấy phần nguyên A cho UCLN
mau_moi = B // UCLN     # Chia lấy phần nguyên B cho UCLN
print(f"Phan so rut gon: {tu_moi}/{mau_moi}")   # 2/3
```

=> Kết quả sau khi chạy chương trình:

```
Phan so rut gon: 2/3
```

1.2. Code C++:

```cpp
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
using namespace std;
// Hàm tìm UCLN của 2 số a, b
long long tim_ucln(long long a, long long b)
{
    // Đảm bảo làm việc với giá trị tuyệt đối (UCLN luôn dương)
    a = abs(a);
    b = abs(b);
    // Thuật toán Euclid lặp
    long long temp;
    while (b != 0)
    {
        temp = a % b;    // temp = so du (r)
        a = b;           // a = so chia (b)
        b = temp;        // b = so du moi (r)
    }
    return a;   // Khi b = 0, UCLN là a
}
// Hàm chính để kiểm tra
int main()
{
    int A = 12;
    int B = 18;
    int UCLN = tim_ucln(A, B);
    int tu_moi = A / UCLN;  // Chia lấy phần nguyên A cho UCLN
    int mau_moi = B / UCLN; // Chia lấy phần nguyên B cho UCLN
    cout << "Phan so rut gon: " << tu_moi << "/" << mau_moi << endl;    // 2/3
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
Phan so rut gon: 2/3
```

2. Bài 2. Kiểm tra hai số nguyên tố cùng nhau.

- **Mô tả:** Nhập hai số **A** và **B**. Kiểm tra xem chúng có phải là **nguyên tố cùng nhau** hay không.
- **Logic:** Hai số **A**, **B** là nguyên tố cùng nhau khi **UCLN(A, B) = 1**.

2.1. Code Python:

```python
# Hàm tìm UCLN của 2 số a, b
def tim_ucln(a, b):
    # Đảm bảo làm việc với giá trị tuyệt đối (UCLN luôn dương)
    a, b = abs(a), abs(b)
    # Thuật toán Euclid lặp
    while b:
        a, b = b, a % b
    return a
# Kiểm tra
A = 15
B = 7
if tim_ucln(A, B) == 1:
    print(f"{A} va {B} nguyen to cung nhau.")
else:
    print(f"{A} va {B} KHONG nguyen to cung nhau.")
```

=> Kết quả sau khi chạy chương trình:

```
15 va 7 nguyen to cung nhau.
```

2.2. Code C++:

```cpp
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
using namespace std;
// Hàm tìm UCLN của 2 số a, b
long long tim_ucln(long long a, long long b)
{
    // Đảm bảo làm việc với giá trị tuyệt đối (UCLN luôn dương)
    a = abs(a);
    b = abs(b);
    // Thuật toán Euclid lặp
    long long temp;
    while (b != 0)
    {
        temp = a % b;    // temp = so du (r)
        a = b;           // a = so chia (b)
        b = temp;        // b = so du moi (r)
    }
    return a;    // Khi b = 0, UCLN là a
}
// Hàm chính để kiểm tra
int main()
{
    int A = 15;
    int B = 7;
    if (tim_ucln(A, B) == 1)
    {
        cout << A << " va " << B << " nguyen to cung nhau." << endl;
    }
    else
    {
        cout << A << " va " << B << " KHONG nguyen to cung nhau." << endl;
    }
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
15 va 7 nguyen to cung nhau.
```

VI. Tổng Kết & Dặn Dò:

1. Củng cố chính:

- **Thuật toán Euclid** là nền tảng để tìm **UCLN** và **BCNN**.
- **UCLN** được tính bằng **while (b != 0): a, b = b, a % b**.
- **BCNN** được tính qua **UCLN** bằng công thức:

  BCNN(a, b) = |a * b| / UCLN(a, b)
- Luôn sử dụng **long long** trong **C++** khi tính **a * b** để tránh tràn số.

2. Dặn dò: Hoàn thành bài tập về nhà. Buổi tiếp theo chúng ta sẽ đi tìm hiểu kĩ hơn về **Số nguyên tố – Sàng Eratosthenes**.

BÀI 8. SỐ NGUYÊN TỐ - SÀNG NGUYÊN TỐ

I. Mục Tiêu:

- **Kiến thức:** Học sinh (**HS**) hiểu được tầm quan trọng của việc kiểm tra và tìm số nguyên tố hiệu quả. Nắm vững nguyên lý và logic hoạt động của thuật toán **Sàng Eratosthenes**.
- **Kỹ năng:** HS thành thạo cách cài đặt Sàng Eratosthenes để tạo danh sách số nguyên tố hoặc mảng đánh dấu các số nguyên tố/hợp số trong một phạm vi lớn ($N \leq 10^6$).
- **Tư duy:** Phân tích được độ phức tạp thời gian O(n log log n)) của Sàng so với kiểm tra nguyên tố thông thường (O(sqrt(n)).

II. Phân Bổ Thời Gian (90 Phút):

Hoạt Động	Nội Dung	Thời gian (Phút)
Khởi động	Ôn tập UCLN/BCNN & Nhắc lại kiểm tra SNT cơ bản	10
Lý thuyết 1	Tại sao cần Sàng? Nguyên lý Sàng Eratosthenes	15
Thực hành 1	Bài 1: Cài đặt Sàng cơ bản (max_N = 100)	30
Thực hành 2	Bài 2: Ứng dụng Sàng (Đếm số nguyên tố trong đoạn)	25
Tổng kết	Bài tập về nhà, Củng cố	10
Tổng cộng		**90**

III. Lý Thuyết: Sàng Eratosthenes (25 phút)

1. Nhắc lại Kiểm tra Số Nguyên Tố Cơ bản:

- Trước đây, để kiểm tra số nguyên tố N, ta dùng phương pháp lặp:

  Độ phức tạp: O(sqrt(N)).
- **Vấn đề:** Nếu cần kiểm tra 10^5 số, mỗi số có giá trị 10^6, tổng thời gian là quá lớn.

2. Nguyên lý Sàng Eratosthenes:

Sàng Eratosthenes là một thuật toán hiệu quả để tìm **tất cả** số nguyên tố trong một phạm vi nhất định (từ 2 đến N).

- **Bước 1. Khởi tạo:** Tạo một mảng is_prime[0..N] và gán tất cả là True (hoặc 1), ngoại trừ 0 và 1.

- **Bước 2. Duyệt:** Bắt đầu từ số p = 2.
- **Bước 3. Đánh dấu:** Nếu p chưa bị đánh dấu (tức là is_prime[p] là True), thì p là một số nguyên tố.
- **Bước 4. Loại bỏ bội số:** Đánh dấu tất cả các bội số của p là False (hoặc 0), bắt đầu từ p^2: p^2, $p^2 + p$, $p^2 + 2p$, …
- **Bước 5. Lặp:** Tăng p lên 1 và lặp lại bước 3, 4.
- **Tại sao bắt đầu từ p^2?** Vì các bội số nhỏ hơn p^2 (ví dụ: 2p, 3p, …) đã bị loại bỏ bởi các số nguyên tố nhỏ hơn (2, 3, …).
- **Độ phức tạp:** O(N log log N). Cực kỳ nhanh!

IV. Thực Hành Kèm Code Mẫu:

1. Bài 1. Cài đặt Sàng Eratosthenes cơ bản (N = 100). (30 phút)

- **Yêu cầu:** Viết hàm **sieve(N)** tạo ra một mảng đánh dấu các số nguyên tố/hợp số đến N. In ra tất cả các số nguyên tố tìm được.

1.1. Code Python:

```python
MAX_N = 100
def sieve(N):
    # 1. Khoi tao: Danh sach boolean (True = SNT)
    is_prime = [True] * (N + 1)
    is_prime[0] = is_prime[1] = False
    # 2. Duyet tu 2 den sqrt(N)
    p = 2
    while p * p <= N:
        if is_prime[p]:
            # 3. Loai bo boi so, bat dau tu p*p
            for i in range(p * p, N + 1, p):
                is_prime[i] = False
        p = p + 1
    # 4. Ghi ra danh sach cac so nguyen to vua tim duoc
    primes = []
    n = len(is_prime)
    # Duyệt mảng is_prime để tìm SNT đã được đánh dấu
    for i in range(n):
        is_p = is_prime[i]
        if is_p:
            primes.append(i)
    return primes # Trả về danh sách SNT sau khi Sàng hoàn tất
print(f"SNT < 100: {sieve(MAX_N)}")
```

=> Kết quả sau khi chạy chương trình:

```
SNT < 100: [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97]
```

- Giải thích Code Python:

+ **Khởi tạo:** Dùng **[True] * (N + 1)** nhanh chóng tạo mảng **boolean**.
+ **Vòng lặp:** Điều kiện lặp **p * p <= N** là chìa khóa để đảm bảo hiệu suất.
+ **Loại bỏ:** Dùng **range(start, end, step)** để nhảy qua các bội số.

1.2. Code C++:

```cpp
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
using namespace std;
int MAX_N = 100;
vector<int> sieve(int N)
{
    // 1. Khoi tao: Danh sach boolean (True = SNT)
```

```
    vector<bool> is_prime(N + 1, true);
    is_prime[0] = false;
    is_prime[1] = false;
    // 2. Duyet tu 2 den sqrt(N)
    int p = 2;
    while (p * p <= N)
    {
        if (is_prime[p])
        {
            // 3. Loai bo boi so, bat dau tu p*p
            for (int i = p * p; i <= N; i += p)
            {
                is_prime[i] = false;
            }
        }
        p = p + 1;
    }
    // 4. Ghi ra danh sach cac so nguyen to vua tim duoc
    vector<int> primes;
    int n = is_prime.size();
    // Duyet mang is_prime de tim SNT da duoc danh dau
    for (int i = 0; i < n; i++)
    {
        bool is_p = is_prime[i];
        if (is_p)
        {
            primes.push_back(i);
        }
    }
    return primes; // Tra ve danh sach SNT sau khi Sang hoan tat
}
int main()
{
    vector<int> primes = sieve(MAX_N);
    cout << "SNT < 100: [";
    for (int i = 0; i < primes.size(); i++)
    {
        cout << primes[i];
        if (i + 1 < primes.size()) cout << ", ";
    }
    cout << "]";
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
SNT < 100: [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97]
```

- Giải thích Code C++:

+ **Khởi tạo:** Dùng **vector<bool>** (có cơ chế tối ưu hóa bộ nhớ đặc biệt).

+ **Loại bỏ:** Vòng **for** lồng bên trong sử dụng **i += p** để nhảy bội số.

+ **Lưu ý:** Nếu **N** rất lớn, cần dùng **long long** cho phép nhân **p * p** trong điều kiện vòng lặp để tránh tràn số.

2. Bài 2. Ứng dụng Sàng (Đếm số nguyên tố trong đoạn). (25 phút)

- **Yêu cầu:** Cho L và R. Đếm số lượng số nguyên tố trong đoạn [L, R].

- **Logic:** Chạy Sàng đến **max(R)**, sau đó chỉ cần duyệt mảng **is_prime** từ L đến R và đếm những vị trí có giá trị True.

2.1. Code Python:

```python
MAX_R = 1000
def count_primes_in_range(L, R):
    # Chay Sang den R
    is_prime = [True] * (R + 1)
    is_prime[0] = is_prime[1] = False
    p = 2
    while p * p <= R:
        if is_prime[p]:
            for i in range(p * p, R + 1, p):
                is_prime[i] = False
        p += 1
    # Dem so SNT trong doan [L, R]
    count = 0
    for i in range(L, R + 1):
        if is_prime[i]:
            count += 1
    return count
L = 10
R = 50
print(f"So luong SNT trong [{L}, {R}]: {count_primes_in_range(L, R)}") # 11 so\n
```

=> Kết quả sau khi chạy chương trình:

```
So luong SNT trong [10, 50]: 11
```

- Giải thích Code Python:

+ **Tái sử dụng:** Toàn bộ logic Sàng được đặt bên trong hàm.

+ **Đếm:** Sau khi sàng xong, ta chỉ cần lặp từ L đến R và cộng dồn. Việc này chỉ mất thời gian O(R-L), rất nhanh.

2.2. Code C++:

```cpp
#include <iostream>         // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>    // Thư viện tổng hợp
using namespace std;
int MAX_R = 1000;
int count_primes_in_range(int L, int R)
{
    // Chay Sang den R
    vector<bool> is_prime(R + 1, true);
    is_prime[0] = false;
    is_prime[1] = false;
    int p = 2;
    while (p * p <= R)
    {
        if (is_prime[p])
        {
            for (int i = p * p; i <= R; i += p)
            {
                is_prime[i] = false;
            }
        }
        p += 1;
    }
    // Dem so SNT trong doan [L, R]
    int count = 0;
    for (int i = L; i <= R; i++)
    {
        if (is_prime[i])
```

```
        {
            count += 1;
        }
    }
    return count;
}
int main()
{
    int L = 10;
    int R = 50;
    cout << "So luong SNT trong [" << L << ", " << R << "]: "
         << count_primes_in_range(L, R);  // 11 so

    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
So luong SNT trong [10, 50]: 11
```

- Giải thích Code C++:

+ **Tái sử dụng:** Toàn bộ logic Sàng được đặt bên trong hàm.

+ **Đếm:** Sau khi sàng xong, ta chỉ cần lặp từ L đến R và cộng dồn. Việc này chỉ mất thời gian O(R-L), rất nhanh.

V. Bài Tập Về Nhà Kèm Gợi Ý Code:

1. Bài 1. Tìm số nguyên tố thứ K.

- **Mô tả:** Nhập số nguyên K. Tìm và in ra số nguyên tố thứ K trong dãy số tự nhiên. (Ví dụ: K = 5, kết quả là 11 vì dãy là 2, 3, 5, 7, 11, 13, …).

- **Logic:** Chạy Sàng đến một giới hạn đủ lớn (ví dụ 10^5), sau đó duyệt qua mảng is_prime và đếm. Dừng lại khi tìm thấy số thứ K.

1.1. Code Python:

```
# Giả thiết K <= 10000 -> Max SNT can tim < 10^5
MAX_LIMIT = 130000
def find_kth_prime(K):
    if K <= 0:
        return -1
    is_prime = [True] * MAX_LIMIT
    is_prime[0] = is_prime[1] = False
    # ----------------------------------------------
    # THỰC HIỆN SÀNG ERATOSTHENES
    # ----------------------------------------------
    p = 2
    while p * p < MAX_LIMIT:
        if is_prime[p]:
            for i in range(p * p, MAX_LIMIT, p):
                is_prime[i] = False
        p += 1
    # ----------------------------------------------
    # ĐẾM VÀ TÌM SỐ NGUYÊN TỐ THỨ K
    # ----------------------------------------------
    count = 0
    for i in range(2, MAX_LIMIT):
        if is_prime[i]:
            count += 1
            if count == K:
                return i # Tìm thấy SNT thứ K
    # Trả về -1 nếu vòng lặp kết thúc (tức là không đủ K số nguyên tố)
```

```
    return -1
print(f"SNT thu 5 la: {find_kth_prime(5)}") # Kết quả: 11
```

=> Kết quả sau khi chạy chương trình:

```
SNT thu 5 la: 11
```

1.2. Code C++:

```
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
using namespace std;
// Gia thiet K <= 10000 -> Max SNT can tim < 10^5
int MAX_LIMIT = 130000;
int find_kth_prime(int K)
{
    if (K <= 0)
    {
        return -1;
    }
    vector<bool> is_prime(MAX_LIMIT, true);
    is_prime[0] = false;
    is_prime[1] = false;
    // ----------------------------------------------
    // THUC HIEN SANG ERATOSTHENES
    // ----------------------------------------------
    int p = 2;
    while (p * p < MAX_LIMIT)
    {
        if (is_prime[p])
        {
            for (int i = p * p; i < MAX_LIMIT; i += p)
            {
                is_prime[i] = false;
            }
        }
        p += 1;
    }
    // ----------------------------------------------
    // DEM VA TIM SO NGUYEN TO THU K
    // ----------------------------------------------
    int count = 0;
    for (int i = 2; i < MAX_LIMIT; i++)
    {
        if (is_prime[i])
        {
            count += 1;
            if (count == K)
            {
                return i; // Tim thay SNT thu K
            }
        }
    }
    // Tra ve -1 neu vong lap ket thuc (tuc la khong du K so nguyen to)
    return -1;
}
int main()
{
    cout << "SNT thu 5 la: " << find_kth_prime(5); // Ket qua: 11
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
SNT thu 5 la: 11
```

2. Bài 2. Phân tích thừa số nguyên tố (sử dụng Sàng).

- **Mô tả:** Nhập số N. In ra các thừa số nguyên tố của N.
- **Logic:** Sau khi Sàng, ta có danh sách các SNT. Duyệt qua danh sách SNT đó. Nếu N chia hết cho SNT, in SNT đó ra và chia N cho SNT cho đến khi N không còn chia hết nữa. Lặp lại cho đến khi N = 1.

2.1. Code Python:

```python
import math # Thư viện để tính căn bậc 2
# Giới hạn tối đa cho Sàng (ví dụ: N <= 10^6)
MAX_LIMIT = 1000000
def sieve_to_get_primes(limit):
    # Sàng Eratosthenes để tạo danh sách SNT đến limit.
    if limit < 2:
        return []
    is_prime = [True] * (limit + 1)
    is_prime[0] = is_prime[1] = False
    p = 2
    while p * p <= limit:
        if is_prime[p]:
            for i in range(p * p, limit + 1, p):
                is_prime[i] = False
        p += 1
    # Tạo danh sách SNT từ mảng boolean
    primes = []
    n = len(is_prime)
    # Duyệt mảng is_prime để tìm SNT đã được đánh dấu
    for i in range(n):
        is_p = is_prime[i]
        if is_p:
            primes.append(i)
    return primes
def prime_factorization(N):
    # Phân tích N thành các thừa số nguyên tố bằng cách sử dụng danh sách SNT đã tạo
bằng Sàng.
    if N <= 1:
        print("Khong co thua so nguyen to.")
        return
    original_N = N
    factors = [] # Danh sách lưu các thừa số nguyên tố
    # Tạo danh sách SNT chỉ cần đến căn bậc hai của MAX_LIMIT
    # Vì thuật toán phân tích chỉ cần SNT đến sqrt(N)
    primes_list = sieve_to_get_primes(int(math.sqrt(MAX_LIMIT)) + 1)
    # 1. Duyệt qua danh sách SNT đã sàng
    for p in primes_list:
        if p * p > N:
            break # Dừng nếu SNT đã lớn hơn căn bậc hai của N hiện tại
        # 2. Nếu N chia hết cho p (p là SNT)
        while N % p == 0:
            factors.append(p) # In/lưu thừa số nguyên tố p
            N //= p           # Chia N cho p (lặp lại cho đến khi không chia hết nữa)
    # 3. Xử lý phần còn lại: Nếu N > 1 sau vòng lặp, N chính là một SNT lớn
    if N > 1:
        factors.append(N)
    print(f"Thua so nguyen to cua {original_N}: {factors}")
```

```
# ----------------- CHẠY THỬ -----------------
N1 = 100
prime_factorization(N1)  # Kết quả: [2, 2, 5, 5]
N2 = 34567
prime_factorization(N2)  # Kết quả: [34567] (vì 34567 là SNT)
```

=> Kết quả sau khi chạy chương trình:

```
Thua so nguyen to cua 100: [2, 2, 5, 5]
Thua so nguyen to cua 34567: [13, 2659]
```

2.2. Code C++:

```
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
#include <cmath>             // Thư viện tính căn bậc 2
using namespace std;
// Gioi han toi da cho Sang (vi du: N <= 10^6)
int MAX_LIMIT = 1000000;
vector<int> sieve_to_get_primes(int limit)
{
    // Sang Eratosthenes de tao danh sach SNT den limit.
    if (limit < 2)
    {
        return {};
    }
    vector<bool> is_prime(limit + 1, true);
    is_prime[0] = false;
    is_prime[1] = false;
    int p = 2;
    while (p * p <= limit)
    {
        if (is_prime[p])
        {
            for (int i = p * p; i <= limit; i += p)
            {
                is_prime[i] = false;
            }
        }
        p += 1;
    }
    // Tao danh sach SNT tu mang boolean
    vector<int> primes;
    int n = is_prime.size();
    // Duyet mang is_prime de tim SNT da duoc danh dau
    for (int i = 0; i < n; i++)
    {
        bool is_p = is_prime[i];
        if (is_p)
        {
            primes.push_back(i);
        }
    }
    return primes;
}
void prime_factorization(long long N)
{
    // Phan tich N thanh cac thua so nguyen to bang cach su dung danh sach SNT da
tao bang Sang.
    if (N <= 1)
    {
```

```
        cout << "Khong co thua so nguyen to." << endl;
        return;
    }
    long long original_N = N;
    vector<long long> factors; // Danh sach luu cac thua so nguyen to
    // Tao danh sach SNT chi can den can bac hai cua MAX_LIMIT
    // Vi thuat toan phan tich chi can SNT den sqrt(N)
    vector<int> primes_list = sieve_to_get_primes((int)sqrt(MAX_LIMIT) + 1);
    // 1. Duyet qua danh sach SNT da sang
    for (int p : primes_list)
    {
        if ((long long)p * p > N)
        {
            break; // Dung neu SNT da lon hon can bac hai cua N hien tai
        }
        // 2. Neu N chia het cho p (p la SNT)
        while (N % p == 0)
        {
            factors.push_back(p); // In/luu thua so nguyen to p
             N /= p;                // Chia N cho p (lap lai cho den khi khong chia
het nua)
        }
    }
    // 3. Xu ly phan con lai: Neu N > 1 sau vong lap, N chinh la mot SNT lon
    if (N > 1)
    {
        factors.push_back(N);
    }
    // In ket qua
    cout << "Thua so nguyen to cua " << original_N << ": [";
    for (int i = 0; i < factors.size(); i++) {
        cout << factors[i];
        if (i + 1 < factors.size()) cout << ", ";
    }
    cout << "]" << endl;
}
// ----------------- CHAY THU -----------------
int main()
{
    long long N1 = 100;
    prime_factorization(N1);  // Ket qua: [2, 2, 5, 5]
    long long N2 = 34567;
    prime_factorization(N2);  // Ket qua: [13, 2659]
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
Thua so nguyen to cua 100: [2, 2, 5, 5]
Thua so nguyen to cua 34567: [13, 2659]
```

VI. Tổng Kết & Dặn Dò:

1. Củng cố chính:

- **Sàng Eratosthenes** là phương pháp tối ưu nhất để tìm **toàn bộ** SNT trong một phạm vi.
- Luôn bắt đầu loại bỏ bội số từ p^2 để tránh lặp lại công việc.
- Duyệt vòng ngoài chỉ cần đến **sqrt(N)**.

2. Dặn dò: Hoàn thành bài tập về nhà. Buổi tiếp theo chúng ta sẽ đi vào các thuật toán Phân tích thừa số

BÀI 9. PHÂN TÍCH THỪA SỐ NGUYÊN TỐ

I. Mục Tiêu:

- **Kiến thức:** Học sinh (**HS**) hiểu định nghĩa và tầm quan trọng của việc phân tích thừa số nguyên tố.
- **Kỹ năng:** HS thành thạo hai phương pháp phân tích thừa số:
  + Phương pháp cơ bản **O(sqrt(N))**.
  + Phương pháp tối ưu hóa bằng cách sử dụng danh sách Số Nguyên Tố (SNT) đã sàng O(Prime_Count + log N)).
- **Tư duy:** Phân biệt được khi nào nên sử dụng Sàng và khi nào nên dùng phương pháp O(sqrt(N)).

II. Phân Bổ Thời Gian (90 Phút):

Hoạt Động	Nội Dung	Thời gian (Phút)
Khởi động	Ôn tập Sàng Eratosthenes & UCLN/BCNN	10
Lý thuyết 1	Định nghĩa & Phương pháp Phân tích thừa số cơ bản	15
Thực hành 1	Bài 1. Cài đặt Phân tích thừa số O(sqrt(N))	25
Thực hành 2	Bài 2. Cài đặt Phân tích thừa số dùng Sàng	30
Tổng kết	Bài tập về nhà, Củng cố	10
Tổng cộng		**90**

III. Lý Thuyết: Các Phương Pháp Phân Tích Thừa Số (25 phút)

1. Định Nghĩa Phân Tích Thừa Số Nguyên Tố

- Mọi số nguyên dương N > 1$đều có thể được biểu diễn duy nhất dưới dạng tích của các lũy thừa của các số nguyên tố:

$$N = p_1^{a_1} \cdot p_2^{a_2} \cdot \cdots \cdot p_k^{a_k}$$

- Ví dụ: $12 = 2^2 . 3^1$

2. Phương pháp 1. Duyệt cơ bản O(sqrt(N)):

- **Nguyên lý:**
  + **Bước 1.** Bắt đầu chia **N** cho số nguyên tố nhỏ nhất (**p = 2**).
  + **Bước 2.** Nếu **N** chia hết cho **p**, ghi nhận **p** là thừa số, chia **N** cho **p**, và lặp lại **Bước 2**.
  + **Bước 3.** Nếu **N** không chia hết cho **p**, tăng **p** lên **1**.
  + **Bước 4.** Lặp lại quá trình cho đến khi **p > sqrt(N)**.
  + **Bước 5.** Nếu sau khi kết thúc vòng lặp, **N > 1**, thì **N** còn lại chính là một thừa số nguyên tố lớn.
- **Ưu điểm:** Không cần Sàng, phù hợp khi chỉ cần phân tích một hoặc vài số **N** lớn.

3. Phương pháp 2. Duyệt bằng Sàng Eratosthenes (Tối ưu)

- **Nguyên lý:**

+ **Bước 1.** Chạy **Sàng Eratosthenes** đến giới hạn (thường là $\sqrt{N_{max}}$ hoặc N_{max}) để có danh sách các SNT nhỏ.

+ **Bước 2.** Duyệt qua danh sách SNT đã sàng đó và thực hiện quá trình chia như Phương pháp 1.

- **Ưu điểm:** Nhanh hơn nếu cần phân tích **nhiều số N** trong một phạm vi.

IV. Thực Hành Kèm Code Tham Khảo:

1. Bài 1. Cài đặt Phân tích thừa số O(sqrt(N)) (25 phút)

- **Yêu cầu:** Viết hàm nhận N và in ra các thừa số nguyên tố của N.

1.1. Code Python:

```python
import math # Thư viện toán học, thường dùng khi chương trình có tính căn bậc 2
def factorize_sqrt_n(N):
    factors = []
    original_N = N
    # Cần xử lý trường hợp N <= 1 để tránh lỗi logic
    if N <= 1:
        print(f"Factors({original_N}): Khong co thua so nguyen to.")
        return
    # 1. Chia cho 2 (xử lý SNT duy nhất là số chẵn)
    while N % 2 == 0:
        factors.append(2)
        N //= 2
    # 2. Chia cho các số lẻ từ 3 đến sqrt(N)
    d = 3
    while d * d <= N:
        while N % d == 0:
            factors.append(d)
            N //= d
        d += 2  # Chỉ xét các số lẻ
    # 3. Xử lý N còn lại (N > 1 là một SNT lớn)
    if N > 1:
        factors.append(N)
    print(f"Factors({original_N}): {factors}")
# Kiểm tra
factorize_sqrt_n(100)       # Kết quả đúng: [2, 2, 5, 5]
factorize_sqrt_n(30030)     # Kết quả đúng: [2, 3, 5, 7, 11, 13]
```

=> Kết quả sau khi chạy chương trình:

```
Factors(100): [2, 2, 5, 5]
Factors(30030): [2, 3, 5, 7, 11, 13]
```

- Giải thích Code Python:

+ **Tối ưu hóa:** Bắt đầu chia cho 2 riêng biệt, sau đó lặp **d += 2** (chỉ xét số lẻ) để giảm một nửa số vòng lặp.

+ **Điều kiện dừng: d * d <= N** (tương đương **d ≤ sqrt(N)**).

1.2. Code C++:

```cpp
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
#include <cmath>             // Thư viện tính căn bậc 2
using namespace std;
// Thu vien toan hoc, thuong dung khi chuong trinh co tinh can bac 2
void factorize_sqrt_n(long long N)
```

```
{
    vector<long long> factors;
    long long original_N = N;
    // Can xu ly truong hop N <= 1 de tranh loi logic
    if (N <= 1) {
        cout << "Factors(" << original_N << "): Khong co thua so nguyen to." << "\n";
        return;
    }
    // 1. Chia cho 2 (xu ly SNT duy nhat la so chan)
    while (N % 2 == 0)
    {
        factors.push_back(2);
        N /= 2;
    }
    // 2. Chia cho cac so le tu 3 den sqrt(N)
    long long d = 3;
    while (d * d <= N)
    {
        while (N % d == 0)
        {
            factors.push_back(d);
            N /= d;
        }
        d += 2; // Chi xet cac so le
    }
    // 3. Xu ly N con lai (N > 1 la mot SNT lon)
    if (N > 1)
    {
        factors.push_back(N);
    }
    cout << "Factors(" << original_N << "): [";
    for (int i = 0; i < factors.size(); i++)
    {
        cout << factors[i];
        if (i + 1 < factors.size()) cout << ", ";
    }
    cout << "]\n";
}
// Kiem tra
int main()
{
    factorize_sqrt_n(100);       // Ket qua dung: [2, 2, 5, 5]
    factorize_sqrt_n(30030);     // Ket qua dung: [2, 3, 5, 7, 11, 13]
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
Factors(100): [2, 2, 5, 5]
Factors(30030): [2, 3, 5, 7, 11, 13]
```

- Giải thích Code C++:

+ **Kiểu dữ liệu:** Sử dụng long long cho N và d để xử lý các số có giá trị lớn ($N \leq 10^{12}$ hoặc 10^{18}).

+ **Tối ưu:** Vẫn dùng **d += 2** để chỉ xét số lẻ.

2. Bài 2. Cài đặt Phân tích thừa số dùng Sàng (30 phút)

- **Yêu cầu:** Kết hợp hàm Sàng Eratosthenes để tạo danh sách SNT, sau đó sử dụng danh sách này để phân tích N. (Giả định $N \leq 10^{16}$).

2.1. Code Python:

```
import math # Thư viện toán học, thường dùng khi chương trình có tính căn bậc 2
# Hằng số xác định giới hạn tối đa cho Sàng.
# Ví dụ: Nếu N tối đa là 10^6, ta chỉ cần Sàng đến căn bậc hai của N (1000).
MAX_SNT_CHECK = 1000
def sieve_for_factors(limit):
    # Thực hiện Sàng Eratosthenes để tạo danh sách SNT đến giới hạn đã cho.
    # Độ phức tạp: O(N log log N).
    # 1. Khởi tạo: Danh sách boolean (True = SNT)
    is_prime = [True] * (limit + 1)
    is_prime[0] = is_prime[1] = False
    # 2. Duyệt từ 2 đến sqrt(limit) và loại bỏ bội số
    for p in range(2, int(math.sqrt(limit)) + 1):
        if is_prime[p]:
            # 3. Loai bo boi so, bat dau tu p*p
            for i in range(p * p, limit + 1, p):
                is_prime[i] = False
    # 4. Ghi ra danh sách các số nguyên tố (Chỉ thực hiện sau khi Sàng xong)
    primes = []
    n = len(is_prime)
    for i in range(n):
        if is_prime[i]:
            primes.append(i)
    return primes
# ----------------------------------------------------
# BƯỚC 1: TIỀN XỬ LÝ - CHẠY SÀNG MỘT LẦN DUY NHẤT
# ----------------------------------------------------
# Lưu danh sách SNT đã sàng vào biến toàn cục PRIME_LIST
PRIME_LIST = sieve_for_factors(MAX_SNT_CHECK)
def factorize_with_sieve(N):
    # Phân tích số N thành các thừa số nguyên tố bằng cách sử dụng PRIME_LIST.
    # Độ phức tạp: O(Số lượng SNT nhỏ hơn sqrt(N) + log N).
    factors = []
    original_N = N
    if N <= 1:
        print(f"Factors({original_N}): Khong co thua so nguyen to.")
        return
    # 1. Duyệt qua danh sách SNT đã sàng (PRIME_LIST)
    for p in PRIME_LIST:
        # Tối ưu: Nếu SNT hiện tại (p) lớn hơn căn bậc hai của N, ta dừng lại
        if p * p > N:
            break
        # 2. Chia N cho p liên tục nếu chia hết
        while N % p == 0:
            factors.append(p)
            N //= p
    # 3. Xử lý N còn lại (N > 1 là một SNT lớn hơn MAX_SNT_CHECK)
    if N > 1:
        factors.append(N)
    print(f"Factors({original_N}): {factors}")
# ----------------------------------------------------
# CHẠY THỬ CHƯƠNG TRÌNH
# ----------------------------------------------------
print("--- Phan tich Thua so Nguyen to ---")
# N = 10^6: chỉ có ước 2 và 5 (đều nhỏ hơn 1000)
factorize_with_sieve(1000000)
# N = 30030: Có các SNT nhỏ
```

```
factorize_with_sieve(30030)
# N = 999999: Có ước lớn
factorize_with_sieve(999999)
# N = 999000000: Có ước 333 (là SNT lớn hơn 1000)
factorize_with_sieve(999000000)
```

=> Kết quả sau khi chạy chương trình:

```
Factors(1000000): [2, 2, 2, 2, 2, 2, 5, 5, 5, 5, 5, 5]
Factors(30030): [2, 3, 5, 7, 11, 13]
Factors(999999): [3, 3, 3, 7, 11, 13, 37]
Factors(999000000): [2, 2, 2, 2, 2, 2, 3, 3, 3, 5, 5, 5, 5, 5, 5, 37]
```

- Giải thích Code Python:

+ **Tối ưu:** Ta chỉ cần Sàng đến sqrt(N_max) = 1000. Việc duyệt qua PRIME_LIST (chỉ có 168 số) sẽ nhanh hơn rất nhiều so với việc duyệt d+=2 đến 1000.

+ **Tiền xử lý:** Sàng chỉ chạy **một lần** trước khi gọi factorize.

2.1. Code C++:

```
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
#include <cmath>              // Thu vien toan hoc, thuong dung khi chuong trinh co
tinh can bac 2
using namespace std;
// Hang so xac dinh gioi han toi da cho Sang.
// Vi du: Neu N toi da la 10^6, ta chi can Sang den can bac hai cua N (1000).
const int MAX_SNT_CHECK = 1000;
vector<int> sieve_for_factors(int limit)
{
    // Thuc hien Sang Eratosthenes de tao danh sach SNT den gioi han da cho.
    // Do phuc tap: O(N log log N).
    // 1. Khoi tao: Danh sach boolean (true = SNT)
    vector<bool> is_prime(limit + 1, true);
    is_prime[0] = false;
    is_prime[1] = false;
    // 2. Duyet tu 2 den sqrt(limit) va loai bo boi so
    for (int p = 2; p <= (int)sqrt(limit); p++)
    {
        if (is_prime[p])
        {
            // 3. Loai bo boi so, bat dau tu p*p
            for (int i = p * p; i <= limit; i += p)
            {
                is_prime[i] = false;
            }
        }
    }
    // 4. Ghi ra danh sach cac so nguyen to (Chi thuc hien sau khi Sang xong)
    vector<int> primes;
    for (int i = 0; i <= limit; i++)
    {
        if (is_prime[i]) primes.push_back(i);
    }

    return primes;
}
// ----------------------------------------------------
// BUOC 1: TIEN XU LY - CHAY SANG MOT LAN DUY NHAT
// ----------------------------------------------------
```

```
// Luu danh sach SNT da sang vao bien toan cuc PRIME_LIST
vector<int> PRIME_LIST = sieve_for_factors(MAX_SNT_CHECK);
void factorize_with_sieve(long long N)
{
    // Phan tich so N thanh cac thua so nguyen to bang cach su dung PRIME_LIST.
    // Do phuc tap: O(So luong SNT nho hon sqrt(N) + log N).
    vector<long long> factors;
    long long original_N = N;
    if (N <= 1)
    {
        cout << "Factors(" << original_N << "): Khong co thua so nguyen to.\n";
        return;
    }
    // 1. Duyet qua danh sach SNT da sang (PRIME_LIST)
    for (int p : PRIME_LIST)
    {
        // Toi uu: Neu SNT hien tai (p) lon hon can bac hai cua N, ta dung lai
        if ((long long)p * p > N) break;
        // 2. Chia N cho p lien tuc neu chia het
        while (N % p == 0)
        {
            factors.push_back(p);
            N /= p;
        }
    }
    // 3. Xu ly N con lai (N > 1 la mot SNT lon hon MAX_SNT_CHECK)
    if (N > 1)
    {
        factors.push_back(N);
    }
    cout << "Factors(" << original_N << "): [";
    for (size_t i = 0; i < factors.size(); i++)
    {
        cout << factors[i];
        if (i + 1 < factors.size()) cout << ", ";
    }
    cout << "]\n";
}
// ----------------------------------------------------
// CHAY THU CHUONG TRINH
// ----------------------------------------------------
int main()
{
    cout << "--- Phan tich Thua so Nguyen to ---\n";
    // N = 10^6: chi co uoc 2 va 5 (deu nho hon 1000)
    factorize_with_sieve(1000000);
    // N = 30030: Co cac SNT nho
    factorize_with_sieve(30030);
    // N = 999999: Co uoc lon
    factorize_with_sieve(999999);
    // N = 999000000: Co uoc 333 (la SNT lon hon 1000)
    factorize_with_sieve(999000000);
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
Factors(1000000): [2, 2, 2, 2, 2, 2, 5, 5, 5, 5, 5, 5]
Factors(30030): [2, 3, 5, 7, 11, 13]
Factors(999999): [3, 3, 3, 7, 11, 13, 37]
Factors(999000000): [2, 2, 2, 2, 2, 2, 3, 3, 3, 5, 5, 5, 5, 5, 5, 37]
```

- Giải thích Code C++:

+ **Tính toán: p * p > N** trong C++ phải cẩn thận với kiểu dữ liệu.

+ **Ưu điểm:** Vòng lặp **for** duyệt qua **primes_list** chỉ mất thời gian **rất ít** (chỉ **168** lần lặp nếu **MAX_SNT_CHECK =1000**).

V. Bài Tập Về Nhà Kèm Gợi Ý Code:

1. Bài 1. Đếm số lượng ước của N.

- Mô tả: Nhập số N. Đếm tổng số lượng ước của N.

- Logic:

Nếu

$$N = p_1^{a_1} \cdot p_2^{a_2} \cdot \dots \cdot p_k^{a_k},$$

thì số lượng ước là:

$$(a_1 + 1)(a_2 + 1)\dots(a_k + 1).$$

- Gợi ý: Sử dụng factorize_sqrt_n (Bài 1), nhưng thay vì append thừa số, ta đếm lũy thừa (a_i) của mỗi SNT và áp dụng công thức.

1.1. Code Python:

```python
import math # Thư viện toán học, thường dùng khi chương trình có tính căn bậc 2
def count_divisors(N):
    # Tính tổng số lượng ước của N bằng phương pháp phân tích thừa số nguyên tố O(sqrt(N)).
    if N <= 0:
        return 0
    if N == 1:
        return 1
    # Khởi tạo tổng số ước ban đầu (bắt đầu bằng 1)
    total_divisors = 1
    # Số N tạm thời để tính toán
    temp_N = N
    # 1. Xử lý thừa số 2
    count_2 = 0
    while temp_N % 2 == 0:
        count_2 += 1 # Đếm lũy thừa (a_i)
        temp_N //= 2
    # Áp dụng công thức: (a_1 + 1)
    if count_2 > 0:
        total_divisors *= (count_2 + 1)
    # 2. Xử lý các thừa số lẻ từ 3 đến sqrt(N)
    d = 3
    while d * d <= temp_N:
        count_d = 0
        while temp_N % d == 0:
            count_d += 1 # Đếm lũy thừa (a_i)
            temp_N //= d
        # Áp dụng công thức: * (a_i + 1)
        if count_d > 0:
            total_divisors *= (count_d + 1)
        d += 2 # Chỉ xét các số lẻ
    # 3. Xử lý phần còn lại (nếu temp_N > 1)
```

```
    # Nếu temp_N > 1, nó là một thừa số nguyên tố lớn với lũy thừa a_k = 1
    if temp_N > 1:
        # Áp dụng công thức: (1 + 1) = 2
        total_divisors *= 2
    return total_divisors
# ----------------- CHẠY THỬ -----------------
N1 = 12     # Phân tích: 2^2 * 3^1. Số ước: (2+1)*(1+1) = 6 (Ước: 1, 2, 3, 4, 6, 12)
N2 = 100    # Phân tích: 2^2 * 5^2. Số ước: (2+1)*(2+1) = 9
N3 = 30030  # Phân tích: 2*3*5*7*11*13. Số ước: 2^6 = 64
N4 = 7      # Số nguyên tố
N5 = 1      # Trường hợp đặc biệt
print(f"N = {N1}, Số lượng ước: {count_divisors(N1)}")
print(f"N = {N2}, Số lượng ước: {count_divisors(N2)}")
print(f"N = {N3}, Số lượng ước: {count_divisors(N3)}")
print(f"N = {N4}, Số lượng ước: {count_divisors(N4)}")
print(f"N = {N5}, Số lượng ước: {count_divisors(N5)}")
```

=> Kết quả sau khi chạy chương trình:

```
N = 12, Số lượng ước: 6
N = 100, Số lượng ước: 9
N = 30030, Số lượng ước: 64
N = 7, Số lượng ước: 2
N = 1, Số lượng ước: 1
```

1.2. Code C++:

```
#include <iostream>         // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>    // Thư viện tổng hợp
#include <cmath>             // Thu vien toan hoc, thuong dung khi chuong trinh co
tinh can bac 2
using namespace std;
int count_divisors(long long N)
{
    // Tinh tong so luong uoc cua N bang phuong phap phan tich thua so nguyen to
O(sqrt(N)).
    if (N <= 0)
        return 0;
    if (N == 1)
        return 1;
    // Khoi tao tong so uoc ban dau (bat dau bang 1)
    long long total_divisors = 1;
    // So N tam thoi de tinh toan
    long long temp_N = N;
    // 1. Xu ly thua so 2
    int count_2 = 0;
    while (temp_N % 2 == 0)
    {
        count_2 += 1; // Dem luy thua (a_i)
        temp_N /= 2;
    }
    // Ap dung cong thuc: (a_1 + 1)
    if (count_2 > 0)
    {
        total_divisors *= (count_2 + 1);
    }
    // 2. Xu ly cac thua so le tu 3 den sqrt(N)
    long long d = 3;
    while (d * d <= temp_N)
    {
        int count_d = 0;
```

```
        while (temp_N % d == 0)
        {
            count_d += 1; // Dem luy thua (a_i)
            temp_N /= d;
        }
        // Ap dung cong thuc: * (a_i + 1)
        if (count_d > 0)
        {
            total_divisors *= (count_d + 1);
        }
        d += 2; // Chi xet cac so le
    }
    // 3. Xu ly phan con lai (neu temp_N > 1)
    // Neu temp_N > 1, no la mot thua so nguyen to lon voi luy thua a_k = 1
    if (temp_N > 1)
    {
        // Ap dung cong thuc: (1 + 1) = 2
        total_divisors *= 2;
    }
    return total_divisors;
}
// ----------------- CHAY THU -----------------
int main()
{
    long long N1 = 12;     // Phan tich: 2^2 * 3^1. So uoc: (2+1)*(1+1) = 6 (Uoc: 1,
2, 3, 4, 6, 12)
    long long N2 = 100;    // Phan tich: 2^2 * 5^2. So uoc: (2+1)*(2+1) = 9
    long long N3 = 30030;  // Phan tich: 2*3*5*7*11*13. So uoc: 2^6 = 64
    long long N4 = 7;      // So nguyen to
    long long N5 = 1;      // Truong hop dac biet
    cout << "N = " << N1 << ", So luong uoc: " << count_divisors(N1) << "\n";
    cout << "N = " << N2 << ", So luong uoc: " << count_divisors(N2) << "\n";
    cout << "N = " << N3 << ", So luong uoc: " << count_divisors(N3) << "\n";
    cout << "N = " << N4 << ", So luong uoc: " << count_divisors(N4) << "\n";
    cout << "N = " << N5 << ", So luong uoc: " << count_divisors(N5) << "\n";
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
N = 12, Số lượng ước: 6
N = 100, Số lượng ước: 9
N = 30030, Số lượng ước: 64
N = 7, Số lượng ước: 2
N = 1, Số lượng ước: 1
```

2. Bài 2. Tìm ước lớn nhất là số lẻ.

- **Mô tả:** Nhập số N. Tìm ước lớn nhất của N mà là số lẻ.
- **Logic:** Ước lẻ lớn nhất của N chính là N sau khi đã loại bỏ **tất cả** các thừa số 2.
- **Gợi ý:** Dùng vòng while (N % 2 == 0): N //= 2; để loại bỏ hết thừa số 2. Kết quả N còn lại chính là đáp án.

2.1. Code Python:

```
import math # Thư viện toán học, thường dùng khi chương trình có tính căn bậc 2
def find_largest_odd_divisor(N):
    # Tìm ước lớn nhất là số lẻ của số nguyên dương N.
    if N <= 0:
        return None # Xử lý trường hợp N không phải số nguyên dương
```

```
    original_N = N
    # Logic: Loại bỏ tất cả các thừa số 2 ra khỏi N
    # Nếu N đã là số lẻ, vòng lặp này sẽ không chạy
    while N % 2 == 0:
        N //= 2
    # N còn lại chính là ước lẻ lớn nhất
    print(f"Uoc le lon nhat cua {original_N} la: {N}")
    return N
# ----------------- CHẠY THỬ -----------------
# Ví dụ 1: Số chẵn có nhiều thừa số 2
N1 = 120    # Phân tích: 120 = 2^3 * 3 * 5. Ước lẻ lớn nhất: 3 * 5 = 15
find_largest_odd_divisor(N1)
# Ví dụ 2: Số chẵn chỉ có một thừa số 2
N2 = 18     # Phân tích: 18 = 2 * 3^2. Ước lẻ lớn nhất: 9
find_largest_odd_divisor(N2)
# Ví dụ 3: Số lẻ (ước lẻ lớn nhất chính là nó)
N3 = 45     # Phân tích: 45 = 3^2 * 5. Ước lẻ lớn nhất: 45
find_largest_odd_divisor(N3)
# Ví dụ 4: Lũy thừa của 2
N4 = 32     # Phân tích: 32 = 2^5. Ước lẻ lớn nhất: 1
find_largest_odd_divisor(N4)
```

=> Kết quả sau khi chạy chương trình:

```
Uoc le lon nhat cua 120 la: 15
Uoc le lon nhat cua 18 la: 9
Uoc le lon nhat cua 45 la: 45
Uoc le lon nhat cua 32 la: 1
```

2.2. Code C++:

```
#include <iostream>         // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>    // Thư viện tổng hợp
#include <cmath>             // Thu vien toan hoc, thuong dung khi chuong trinh co
tinh can bac 2
using namespace std;
long long find_largest_odd_divisor(long long N)
{
    // Tim uoc lon nhat la so le cua so nguyen duong N.
    if (N <= 0)
    {
        return -1; // Xu ly truong hop N khong phai so nguyen duong
    }
    long long original_N = N;
    // Logic: Loai bo tat ca cac thua so 2 ra khoi N
    // Neu N da la so le, vong lap nay se khong chay
    while (N % 2 == 0)
    {
        N /= 2;
    }
    // N con lai chinh la uoc le lon nhat
    cout << "Uoc le lon nhat cua " << original_N << " la: " << N << "\n";
    return N;
}
// ----------------- CHAY THU -----------------
int main()
{
    // Vi du 1: So chan co nhieu thua so 2
    long long N1 = 120;   // Phan tich: 120 = 2^3 * 3 * 5. Uoc le lon nhat: 3 * 5 =
15
    find_largest_odd_divisor(N1);
```

```
    // Vi du 2: So chan chi co mot thua so 2
    long long N2 = 18;    // Phan tich: 18 = 2 * 3^2. Uoc le lon nhat: 9
    find_largest_odd_divisor(N2);
    // Vi du 3: So le (uoc le lon nhat chinh la no)
    long long N3 = 45;    // Phan tich: 45 = 3^2 * 5. Uoc le lon nhat: 45
    find_largest_odd_divisor(N3);
    // Vi du 4: Luy thua cua 2
    long long N4 = 32;    // Phan tich: 32 = 2^5. Uoc le lon nhat: 1
    find_largest_odd_divisor(N4);
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
Uoc le lon nhat cua 120 la: 15
Uoc le lon nhat cua 18 la: 9
Uoc le lon nhat cua 45 la: 45
Uoc le lon nhat cua 32 la: 1
```

VI. Tổng Kết & Dặn Dò:

1. Củng cố chính:

- Phương pháp **O(sqrt(N))** đơn giản, hiệu quả cho N đơn lẻ.
- Phương pháp **Dùng Sàng** nhanh hơn nhiều khi cần phân tích nhiều số N trong cùng một phạm vi nhỏ ($N \leq 10^6$).
- Thừa số nguyên tố còn lại N > 1 sau vòng lặp luôn là SNT.

2. Dặn dò: Hoàn thành bài tập về nhà. Buổi tiếp theo chúng ta sẽ chính thức bắt đầu với chủ đề **Số học nâng cao: Đồng dư thức**.

CHỦ ĐỀ 4. SỐ HỌC NÂNG CAO & HÌNH HỌC CƠ BẢN

BÀI 10. ĐỒNG DƯ THỨC

I. Mục Tiêu:

- **Kiến thức:** Học sinh (**HS**) hiểu định nghĩa và các tính chất cơ bản của đồng dư thức. Nắm vững phép toán module cơ bản (cộng, trừ, nhân, lũy thừa).
- **Kỹ năng:** HS thành thạo việc tính toán nhanh giá trị lớn của phép lũy thừa module (ví dụ: A^B (mod M)) bằng thuật toán **Lũy thừa nhanh** (Exponentiation by Squaring).
- **Tư duy:** Áp dụng đồng dư thức để giải quyết các bài toán liên quan đến chu kỳ, tính chữ số cuối cùng và kiểm tra tính chia hết.

II. Phân Bổ Thời Gian (90 Phút):

Hoạt Động	Nội Dung	Thời gian (Phút)
Khởi động	Ôn tập Phân tích thừa số & Lũy thừa cơ bản	10
Lý thuyết 1	Định nghĩa Đồng dư thức và Tính chất cơ bản	15
Thực hành 1	Bài 1. Cài đặt Lũy thừa nhanh (A^B (mod M))	35
Thực hành 2	Bài 2: Ứng dụng: Tính chữ số cuối cùng (Đồng dư (mod 10))	20
Tổng kết	Bài tập về nhà, Củng cố	10
Tổng cộng		**90**

III. Lý Thuyết: Đồng Dư Thức (Modular Arithmetic) (15 phút)

1. Định Nghĩa Đồng Dư Thức:

- Hai số nguyên **a** và **b** được gọi là đồng dư với nhau theo **modulo M** nếu chúng có cùng số dư khi chia cho **M**.

+ **Ký hiệu:**

$$a \equiv b \pmod{M}$$

+ **Ý nghĩa:** a - b chia hết cho M, hoặc a (mod M) = b (mod M).

- Ví dụ:

$17 \equiv 5 \pmod{12}$ vì $17 \pmod{12} = 5$ và $5 \pmod{12} = 5$.

2. Các Tính Chất Cơ Bản:

- Các phép toán cộng, trừ, và nhân đều được bảo toàn trong đồng dư thức:

+ Tính chất Cộng/Trừ:

(a ± b) mod M = ((a (mod M)) ± (b (mod M))) (mod M)

+ **Tính chất Nhân:**

$$(a \cdot b) \pmod{M} = ((a \pmod{M}) \cdot (b \pmod{M})) \pmod{M}$$

+ **Tính chất Lũy thừa:**

$$a^n \pmod{M} = (a \pmod{M})^n \pmod{M}$$

- **Lưu ý quan trọng:** Trong lập trình, khi tính (A . B) (mod M) trong C++, luôn phải ép kiểu (long long)A * B trước khi lấy (mod M) để tránh tràn số (integer overflow) nếu A, B và M là số nguyên lớn.

IV. Thực Hành Kèm Code Tham Khảo:

1. Bài 1. Lũy thừa nhanh (Exponentiation by Squaring) (35 phút)

- **Yêu cầu:** Tính ($base^{exponent}$) (mod modulus).

- **Giải thích yêu cầu:** Ký hiệu này yêu cầu bạn thực hiện ba bước tính toán theo thứ tự:

1. Phép Lũy Thừa ($base^{exponent}$):

+ Ý nghĩa: Tính giá trị của cơ số (base) được nâng lên lũy thừa (exponent).

+ Ví dụ: Nếu base = 2 và exponent = 10, thì $2^{10} = 1024$.

2. Phép Toán Modulo ((mod modulus)):

+ Ý nghĩa: Sau khi tính lũy thừa, lấy kết quả đó chia cho **số chia** (modulus) và tìm **số dư** của phép chia đó.

+ Tác dụng: Phép toán này giữ cho kết quả cuối cùng nằm trong phạm vi từ 0 đến modulus - 1.

3. Kết hợp (Lũy thừa Module):

Khi gộp lại, công thức yêu cầu bạn tìm số dư của **$base^{exponent}$** khi chia cho modulus.

- **Ví dụ Minh Họa:** Giả sử bạn cần tính: (2^{10}) (mod 13)

+ Lũy thừa: $2^{10} = 1024$.

+ **Modulo:** Lấy 1024 chia cho 13:

1024 = 3 x 78 + 10

+ Kết quả: (2^{10}) (mod 13) = 10

- **Ứng dụng trong lập trình:** Trong lập trình thi đấu và mật mã học, phép tính này rất quan trọng khi base và exponent là những số cực lớn (10^{18} hoặc lớn hơn).

+ **Vấn đề:** Khi exponent quá lớn, **base^{exponent}** sẽ là một số khổng lồ không thể lưu trữ được bằng kiểu dữ liệu thông thường.

+ **Giải pháp:** Sử dụng thuật toán **Lũy thừa Nhanh** (O(log(exponent))) để thực hiện các phép nhân modulo trong từng bước, giúp giữ kết quả luôn nhỏ (trong phạm vi modulus) mà vẫn đảm bảo tính chính xác của phép toán đồng dư.

- **Phương pháp cơ bản (lặp B lần):** Là quá chậm ((O(B)) khi B lớn. Ta cần dùng **Lũy thừa nhanh** (O(log B)).

- **Nguyên lý Lũy thừa nhanh:** Dựa trên việc phân tích B thành dạng nhị phân.

+ Nếu B là chẵn: $A^B = (A^2)^{B/2}$

+ Nếu B là lẻ: $A^B = A \cdot A^{B-1}$

1.1. Code Python:

```python
def power(base, exponent, modulus):
    result = 1
    base %= modulus # Rút gọn base truoc
    while exponent > 0:
        # 1. BƯỚC CỘNG GỘP (Chỉ khi exponent lẻ/bit cuối là 1)
        if exponent % 2 == 1:
            # result = (result * base) % modulus
            result = (result * base) % modulus
        # 2. BƯỚC BÌNH PHƯƠNG VÀ DỊCH BIT (BẮT BUỘC MỖI LẦN LẶP)
        # base = (base * base) % modulus
        base = (base * base) % modulus
        # exponent //= 2 (Dich bit sang phai)
        exponent //= 2
    # 3. TRẢ VỀ KẾT QUẢ ĐÚNG VỊ TRÍ (Sau khi vòng lặp kết thúc)
    return result
# Kiem tra: 2^10 mod 13 = 1024 mod 13 = 10
print(f"2^10 mod 13: {power(2, 10, 13)}")
# Kiem tra: 5^100 mod 7 = 4
print(f"5^100 mod 7: {power(5, 100, 7)}")
```

=> Kết quả sau khi chạy chương trình:

```
2^10 mod 13: 10
5^100 mod 7: 2
```

- Giải thích Code Python:

+ **Tốc độ: exponent //= 2** và **base = (base * base) % modulus** là các bước then chốt. Python tự động xử lý số nguyên lớn, nên tràn số không là vấn đề lớn như C++.

+ **Chú ý: Lấy base %= modulus** ở đầu để rút gọn số.

1.2. Code C++:

```cpp
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
using namespace std;
long long power(long long base, long long exponent, long long modulus)
{
    long long result = 1;
    base %= modulus; // Rut gon base truoc
    while (exponent > 0)
    {
        // 1. BUOC CONG GOP (Chi khi exponent le/bit cuoi la 1)
        if (exponent % 2 == 1)
        {
```

```
            // result = (result * base) % modulus
            result = (result * base) % modulus;
        }
        // 2. BUOC BINH PHUONG VA DICH BIT (BAT BUOC MOI LAN LAP)
        // base = (base * base) % modulus
        base = (base * base) % modulus;
        // exponent //= 2 (Dich bit sang phai)
        exponent /= 2;
    }
    // 3. TRA VE KET QUA DUNG VI TRI (Sau khi vong lap ket thuc)
    return result;
}
int main()
{
    // Kiem tra: 2^10 mod 13 = 1024 mod 13 = 10
    cout << "2^10 mod 13: " << power(2, 10, 13) << endl;
    // Kiem tra: 5^100 mod 7 = 4
    cout << "5^100 mod 7: " << power(5, 100, 7) << endl;
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
2^10 mod 13: 10
5^100 mod 7: 2
```

- Giải thích Code C++:

AN TOÀN: Bắt buộc sử dụng **long long** cho tất cả các tham số và biến trung gian (result, base) để đảm bảo phép nhân (base * base) hoặc (result * base) không bị tràn số, đặc biệt khi base và modulus gần 2.10^9.

2. Bài 2. Ứng dụng: Tính chữ số cuối cùng (Đồng dư (mod 10)) (20 phút)

- **Yêu cầu:** Tìm chữ số cuối cùng của phép tính A^B.
- **Logic:** Chữ số cuối cùng của một số X chính là X mod 10. Ta sử dụng hàm power đã viết với modulus = 10.

2.1. Code Python:

```
def power(base, exponent, modulus):
    result = 1
    base = base % modulus
    while exponent > 0:
        if exponent % 2 == 1:
            result = (result * base) % modulus
        base = (base * base) % modulus
        exponent = exponent // 2
    return result
def last_digit_power(A, B):
    if B == 0:
        return 1    # A^0 = 1
    return power(A, B, 10)
# Kiem tra: Chu so cuoi cung cua 3^5
# 3^5 = 243. Ket qua la 3.
print(f"Chu so cuoi cua 3^5: {last_digit_power(3, 5)}")         # 3
# Kiem tra: Chu so cuoi cung cua 123^45
# 123^45 mod 10 = 3^45 mod 10 = 3
print(f"Chu so cuoi cua 123^45: {last_digit_power(123, 45)}")   # 3
```

=> Kết quả sau khi chạy chương trình:

```
Chu so cuoi cua 3^5: 3
Chu so cuoi cua 123^45: 3
```

- Giải thích Code Python:

Rút gọn: Nhờ tính chất đồng dư, A (mod 10) chỉ cần lấy chữ số cuối của A. Hàm **power** đã tự động xử lý base %= 10 nên A lớn không gây vấn đề.

2.2. Code C++:

```
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
using namespace std;
long long power(long long base, long long exponent, long long modulus)
{
    long long result = 1;
    base = base % modulus;
    while (exponent > 0)
    {
        if (exponent % 2 == 1)
        {
            result = (result * base) % modulus;
        }
        base = (base * base) % modulus;
        exponent = exponent / 2;
    }
    return result;
}
long long last_digit_power(long long A, long long B)
{
    if (B == 0)
    {
        return 1;    // A^0 = 1
    }
    return power(A, B, 10);
}
int main()
{
    // Kiem tra: Chu so cuoi cung cua 3^5
    // 3^5 = 243. Ket qua la 3.
    cout << "Chu so cuoi cua 3^5: " << last_digit_power(3, 5) << endl;        // 3
    // Kiem tra: Chu so cuoi cung cua 123^45
    // 123^45 mod 10 = 3^45 mod 10 = 3
    cout << "Chu so cuoi cua 123^45: " << last_digit_power(123, 45) << endl;  // 3
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
Chu so cuoi cua 3^5: 3
Chu so cuoi cua 123^45: 3
```

- Giải thích Code C++:

Lưu ý: Vì kết quả (mod 10) luôn là một chữ số (từ 0 đến 9), ta có thể ép kiểu về **int** cho kết quả cuối cùng nếu muốn, nhưng sử dụng **long long** xuyên suốt vẫn đảm bảo an toàn.

V. Bài Tập Về Nhà Kèm Gợi Ý Code:

1. Bài 1. Tính Fibonacci(N) (mod M).

- Mô tả: Cho N lớn (N ≤ 10^18) và M nhỏ (M ≤ 10^5). Tính số Fibonacci thứ N modulo M.

- **Logic:** Sử dụng **Ma trận Lũy thừa nhanh** (Matrix Exponentiation).

+ Công thức ma trận:

$$\begin{pmatrix} F_{n+1} \\ F_n \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n \begin{pmatrix} F_1 \\ F_0 \end{pmatrix}$$

+ Áp dụng thuật toán Lũy thừa nhanh cho ma trận:

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

- **Gợi ý:** Viết hàm nhân hai ma trận 2 x 2 có lấy (mod M), sau đó viết hàm power_matrix tương tự như power thường.

1.1. Code Python:

```
# Hằng số modulo M sẽ được truyền vào hàm
MODULUS = 100000
def multiply_matrices(A, B, M):
    # Nhân hai ma trận 2x2 (A * B) và lấy modulo M.
    # A = [[a, b], [c, d]]
    # B = [[e, f], [g, h]]
    C = [[0, 0], [0, 0]]
    C[0][0] = (A[0][0] * B[0][0] + A[0][1] * B[1][0]) % M
    C[0][1] = (A[0][0] * B[0][1] + A[0][1] * B[1][1]) % M
    C[1][0] = (A[1][0] * B[0][0] + A[1][1] * B[1][0]) % M
    C[1][1] = (A[1][0] * B[0][1] + A[1][1] * B[1][1]) % M
    return C
# 2. Hàm Lũy Thừa Ma Trận Nhanh (Matrix Exponentiation)
# Tương tự như power thông thường, ta áp dụng thuật toán Lũy thừa nhanh cho phép nhân
ma trận.
# Ma trận chuyển đổi T (Transition Matrix)
T = [[1, 1], [1, 0]]
def power_matrix(N, M):
    # Tính ma trận T^N modulo M, sử dụng thuật toán lũy thừa nhanh.
    # Ma trận kết quả ban đầu là ma trận đơn vị I (Identity Matrix)
    result = [[1, 0], [0, 1]]
    # Ma trận cơ số (Base Matrix)
    base = T
    exponent = N
    while exponent > 0:
        # Nếu exponent lẻ: nhân result với base hiện tại
        if exponent % 2 == 1:
            result = multiply_matrices(result, base, M)
        # Bình phương base cho bước tiếp theo
        base = multiply_matrices(base, base, M)
        # Giảm số mũ đi một nửa
        exponent //= 2
    return result
# 3. Hàm Chính: Tính Fibonacci(N) mod M
def fibonacci_mod_M(N, M):
    # Tính số Fibonacci thứ N modulo M (N >= 1).
    # F0 = 0, F1 = 1.
    if N == 0:
        return 0
    if N == 1:
        return 1
    T_power = power_matrix(N - 1, M)
```

```
    return T_power[0][0]
# ----------------- CHẠY THỬ -----------------
# N lớn (ví dụ N = 10^9) và M nhỏ (ví dụ M = 1000)
N_large = 1000000000
M_small = 1000
# N nhỏ để kiểm tra tính đúng đắn: F10 = 55. 55 mod 100 = 55
N_check = 10
M_check = 100
print(f"F({N_check}) mod {M_check} (F10 = 55): {fibonacci_mod_M(N_check, M_check)}")
# F(10^9) mod 1000
result_large = fibonacci_mod_M(N_large, M_small)
print(f"F({N_large}) mod {M_small}: {result_large}")
```

=> Kết quả sau khi chạy chương trình:

```
F(10) mod 100 (F10 = 55): 55
F(1000000000) mod 1000: 875
```

1.2. Code C++:

```
#include <iostream>         // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>    // Thư viện tổng hợp
using namespace std;
// Ham nhan hai ma tran 2x2 (A * B) va lay modulo M.
void multiply_matrices(long long A[2][2], long long B[2][2], long long C[2][2], long
long M)
{
    // A = [[a, b], [c, d]]
    // B = [[e, f], [g, h]]
    long long c00 = (A[0][0] * B[0][0] + A[0][1] * B[1][0]) % M;
    long long c01 = (A[0][0] * B[0][1] + A[0][1] * B[1][1]) % M;
    long long c10 = (A[1][0] * B[0][0] + A[1][1] * B[1][0]) % M;
    long long c11 = (A[1][0] * B[0][1] + A[1][1] * B[1][1]) % M;
    C[0][0] = c00;
    C[0][1] = c01;
    C[1][0] = c10;
    C[1][1] = c11;
}
// Ma tran chuyen doi T (Transition Matrix)
long long T_mat[2][2] = { {1, 1}, {1, 0} };
// Ham Luy Thua Ma Tran Nhanh (Matrix Exponentiation)
// Tuong tu nhu power thong thuong, ta ap dung thuat toan Luy thua nhanh
void power_matrix(long long N, long long M, long long result[2][2])
{
    // Ma tran ket qua ban dau la ma tran don vi I (Identity Matrix)
    result[0][0] = 1; result[0][1] = 0;
    result[1][0] = 0; result[1][1] = 1;
    // Ma tran co so (Base Matrix)
    long long base[2][2] = { {1, 1}, {1, 0} };
    long long exponent = N;
    while (exponent > 0)
    {
        // Neu exponent le: nhan result voi base hien tai
        if (exponent % 2 == 1)
        {
            long long temp[2][2];
            multiply_matrices(result, base, temp, M);
            result[0][0] = temp[0][0]; result[0][1] = temp[0][1];
            result[1][0] = temp[1][0]; result[1][1] = temp[1][1];
        }
        // Binh phuong base cho buoc tiep theo
```

```
        long long temp2[2][2];
        multiply_matrices(base, base, temp2, M);
        base[0][0] = temp2[0][0]; base[0][1] = temp2[0][1];
        base[1][0] = temp2[1][0]; base[1][1] = temp2[1][1];
        // Giam so mu di mot nua
        exponent /= 2;
    }
}
// Ham Chinh: Tinh Fibonacci(N) mod M
long long fibonacci_mod_M(long long N, long long M)
{
    // Tinh so Fibonacci thu N modulo M (N >= 1).
    // F0 = 0, F1 = 1.
    if (N == 0) return 0;
    if (N == 1) return 1;
    long long T_power[2][2];
    power_matrix(N - 1, M, T_power);
    return T_power[0][0];
}
int main()
{
    // N lon (vi du N = 10^9) va M nho (vi du M = 1000)
    long long N_large = 1000000000;
    long long M_small = 1000;
    // N nho de kiem tra tinh dung dan: F10 = 55. 55 mod 100 = 55
    long long N_check = 10;
    long long M_check = 100;
    cout << "F(" << N_check << ") mod " << M_check << " (F10 = 55): "
         << fibonacci_mod_M(N_check, M_check) << endl;
    // F(10^9) mod 1000
    long long result_large = fibonacci_mod_M(N_large, M_small);
    cout << "F(" << N_large << ") mod " << M_small << ": " << result_large << endl;
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
F(10) mod 100 (F10 = 55): 55
F(1000000000) mod 1000: 875
```

2. Bài 2. Tính chu kỳ module (Pisano Period).

- **Mô tả:** Tìm độ dài chu kỳ của dãy Fibonacci (mod M).
- **Logic:** Dãy Fibonacci (mod M) là tuần hoàn. Ta tìm chu kỳ д(M) bằng cách tìm cặp (F_n, F_{n-1} lặp lại (0, 1).
- **Gợi ý:** Dùng vòng for lặp đơn giản và áp dụng phép (mod M) sau mỗi bước tính $F_{n+1} = (F_n + F_{n-1})$ (mod M).

2.1. Code Python:

```
def pisano_period(M):
    # Tìm độ dài chu kỳ Pisano (pi(M)) của dãy Fibonacci modulo M.
    if M <= 0:
        return 0
    if M == 1:
        return 1 # F(n) mod 1 luôn là 0, chu kỳ là [0], độ dài 1.
    # Khởi tạo hai số Fibonacci đầu tiên modulo M
    # F0 = 0, F1 = 1
    previous = 0
    current = 1
    # Bắt đầu vòng lặp để tìm chu kỳ
```

```
    # Chu kỳ Pisano tối đa là M*M, nên vòng lặp này luôn kết thúc
    for i in range(M * M):
        # F_new = (F_current + F_previous) mod M
        # Ta dùng giá trị tạm thời để cập nhật F_previous và F_current
        temp = (previous + current) % M
        # Cập nhật F_previous -> F_current
        previous = current
        # Cập nhật F_current -> F_new
        current = temp
         # Kiểm tra điều kiện lặp lại: Cặp (F_current, F_previous) có bằng (1, 0)
không?
        # F_current (mới) = 1
        # F_previous (mới) = 0
        if previous == 0 and current == 1:
            # i+1 là độ dài của chu kỳ
            # i là chỉ số của F_n vừa được tính.
            # F_0, F_1, F_2, ..., F_i. i+1 là số phần tử.
            return i + 1
# ---------------- CHẠY THỬ ----------------
# Ví dụ 1: M = 2
# Dãy F(n) mod 2: 0, 1, 1, 0, 1, 1, 0, ...
# Chu kỳ là [0, 1, 1]. Độ dài: 3.
M1 = 2
print(f"Chu kỳ Pisano pi({M1}): {pisano_period(M1)}") # Kết quả: 3
# Ví dụ 2: M = 3
# Dãy F(n) mod 3: 0, 1, 1, 2, 0, 2, 2, 1, 0, 1, ...
# Chu kỳ là [0, 1, 1, 2, 0, 2, 2, 1]. Độ dài: 8.
M2 = 3
print(f"Chu kỳ Pisano pi({M2}): {pisano_period(M2)}") # Kết quả: 8
# Ví dụ 3: M = 10 (Chu kỳ chữ số cuối cùng)
# Pi(10) = 60
M3 = 10
print(f"Chu kỳ Pisano pi({M3}): {pisano_period(M3)}") # Kết quả: 60
```

=> Kết quả sau khi chạy chương trình:

```
F(10) mod 100 (F10 = 55): 55
F(1000000000) mod 1000: 875
```

2.2. Code C++:

```
#include <iostream>         // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>    // Thư viện tổng hợp
using namespace std;
long long pisano_period(long long M)
{
    // Tim do dai chu ky Pisano (pi(M)) cua day Fibonacci modulo M.
    if (M <= 0)
    {
        return 0;
    }
    if (M == 1)
    {
        return 1; // F(n) mod 1 luon la 0, chu ky la [0], do dai 1.
    }
    // Khoi tao hai so Fibonacci dau tien modulo M
    // F0 = 0, F1 = 1
    long long previous = 0;
    long long current = 1;
    // Bat dau vong lap de tim chu ky
    // Chu ky Pisano toi da la M*M, nen vong lap nay luon ket thuc
```

```
    for (long long i = 0; i < M * M; i++)
    {
        // F_new = (F_current + F_previous) mod M
        // Ta dung gia tri tam thoi de cap nhat F_previous va F_current
        long long temp = (previous + current) % M;
        // Cap nhat F_previous -> F_current
        previous = current;
        // Cap nhat F_current -> F_new
        current = temp;
         // Kiem tra dieu kien lap lai: Cap (F_current, F_previous) co bang (1, 0)
khong?
        // F_current (moi) = 1
        // F_previous (moi) = 0
        if (previous == 0 && current == 1)
        {
            // i+1 la do dai cua chu ky
            // i la chi so cua F_n vua duoc tinh.
            // F_0, F_1, F_2, ..., F_i. i+1 la so phan tu.
            return i + 1;
        }
    }
    return 0; // Tranh truong hop should-not-happen
}
int main()
{
    // Vi du 1: M = 2
    // Day F(n) mod 2: 0, 1, 1, 0, 1, 1, 0, ...
    // Chu ky la [0, 1, 1]. Do dai: 3.
    long long M1 = 2;
    cout << "Chu ky Pisano pi(" << M1 << "): " << pisano_period(M1) << endl; // Ket
qua: 3
    // Vi du 2: M = 3
    // Day F(n) mod 3: 0, 1, 1, 2, 0, 2, 2, 1, 0, 1, ...
    // Chu ky la [0, 1, 1, 2, 0, 2, 2, 1]. Do dai: 8.
    long long M2 = 3;
    cout << "Chu ky Pisano pi(" << M2 << "): " << pisano_period(M2) << endl; // Ket
qua: 8
    // Vi du 3: M = 10 (Chu ky chu so cuoi cung)
    // Pi(10) = 60
    long long M3 = 10;
    cout << "Chu ky Pisano pi(" << M3 << "): " << pisano_period(M3) << endl; // Ket
qua: 60
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
F(10) mod 100 (F10 = 55): 55
F(1000000000) mod 1000: 875
```

VI. Tổng Kết & Dặn Dò:

1. Củng cố chính:

- Đồng dư thức cho phép tính toán với số lớn bằng cách giữ chúng trong phạm vi nhỏ.

- Lũy thừa nhanh là thuật toán **bắt buộc** phải sử dụng khi tính A^B (mod M) với B lớn, giảm độ phức tạp từ O(B) xuống O(log B).

- Luôn chú ý đến vấn đề **tràn số** (long long trong C++) khi nhân hai số gần bằng M.

2. Dặn dò: Hoàn thành bài tập về nhà. Buổi tiếp theo chúng ta sẽ đi vào chủ đề **Hàm Euler**.

BÀI 11. HÀM PHI EULER & ĐỊNH LÝ EULER

I. Mục Tiêu:

- **Kiến thức:** Học sinh (**HS**) hiểu định nghĩa Hàm Phi Euler (ø(n)) và các công thức tính toán. Nắm vững Định lý Euler và công thức mở rộng của nó.
- **Kỹ năng:** HS thành thạo việc tính (ø(n) bằng phương pháp phân tích thừa số nguyên tố. Áp dụng Định lý Euler để giảm số mũ trong các phép tính lũy thừa module rất lớn (A^B (mod M)).
- **Tư duy:** Nhận diện các bài toán cần sử dụng Định lý Euler (khi số mũ B rất lớn và B ≥ ø(n))

II. Phân Bổ Thời Gian (90 Phút):

Hoạt Động	Nội Dung	Thời gian (Phút)
Khởi động	Ôn tập Lũy thừa nhanh & Đồng dư	10
Lý thuyết 1	Định nghĩa Hàm Phi Euler (ø(n)) và Công thức	15
Thực hành 1	Bài 1. Cài đặt tính (ø(n)) bằng Phân tích thừa số	25
Lý thuyết 2	Định lý Euler và Định lý Fermat nhỏ	10
Thực hành 2	Bài 2. Ứng dụng Định lý Euler (A^B (mod M) với B rất lớn)	20
Tổng kết	Bài tập về nhà, Củng cố	10
Tổng cộng		**90**

III. Lý Thuyết: Hàm Phi Euler (ø(n)) (25 phút)

1. Định Nghĩa Hàm Phi Euler (ø(n)):

- Hàm Phi Euler, ø(n), là số lượng các số nguyên dương k nhỏ hơn hoặc bằng n và nguyên tố cùng nhau với n (tức là ƯCLN(k, n) = 1).
- **Ví dụ:** ø(10). Các số nguyên tố cùng nhau với 10 và ≤ 10 là: 1, 3, 7, 9.

  => ø(10) = 4

2. Công Thức Tính ø(10)

- Nếu phân tích thừa số nguyên tố của n là:

$$p_1^{a_1} \cdot p_2^{a_2} \cdot \cdots \cdot p_k^{a_k}$$

thì công thức tính ø(n) là:

$$\phi(n) = n \cdot \left(1 - \frac{1}{p_1}\right) \cdot \left(1 - \frac{1}{p_2}\right) \cdot \cdots \cdot \left(1 - \frac{1}{p_k}\right)$$

- **Ví dụ: 10 = 2^1 . 5^1 thì:**

$$\phi(10) = 10 \cdot \left(1 - \frac{1}{2}\right) \cdot \left(1 - \frac{1}{5}\right) = 10 \cdot \frac{1}{2} \cdot \frac{4}{5} = 4$$

IV. Thực Hành Kèm Code Tham Khảo:

1. Bài 1. Cài đặt tính ø(n) (25 phút)

Yêu cầu: Viết hàm tính ø(n) bằng cách phân tích thừa số nguyên tố của n (sử dụng logic O(sqrt(n)).

1.1. Code Python:

```
import math
def euler_phi(n):
    if n == 0:
        return 0
    result = n
    # 1. Xử lý SNT p=2
    if n % 2 == 0:
        # result = result - result / p
        result = result - result // 2
        # Loai bo het thua so 2
        while n % 2 == 0:
            n //= 2
    # 2. Xử lý các SNT lẻ p >= 3
    p = 3
    while p * p <= n:
        if n % p == 0:
            # Ap dung cong thuc: result = result - result / p
            result = result - result // p
            # Loai bo het thua so p
            while n % p == 0:
                n //= p
        # TĂNG p ĐÚNG VỊ TRÍ: Sau khi xử lý xong TẤT CẢ lũy thừa của p,
        # ta mới chuyển sang số lẻ tiếp theo.
        p += 2
    # 3. Neu n con lai la SNT lon (> sqrt ban dau)
    if n > 1:
        result = result - result // n
    return result
print(f"Phi(10) = {euler_phi(10)}") # 4
print(f"Phi(17) = {euler_phi(17)}") # 16
```

=> Kết quả sau khi chạy chương trình:

```
Phi(10) = 4
Phi(17) = 16
```

- Giải thích Code Python:

+ **Tối ưu tính toán:** Thay vì dùng số thực **1 - 1/p**, ta dùng phép toán số nguyên: **n.(1 - 1/p) = n - n/p**. Ta dùng **result = result – result // p** để tránh lỗi số thực.

+ **Lưu ý:** Vòng lặp phải xét **p = 2** và **p += 2** sau **p = 2**.

1.2. Code C++:

```
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
using namespace std;
long long euler_phi(long long n)
{
    if (n == 0)
    {
        return 0;
    }
    long long result = n;
    // 1. Xu ly SNT p=2
```

```
    if (n % 2 == 0)
    {
        // result = result - result / p
        result = result - result / 2;
        // Loai bo het thua so 2
        while (n % 2 == 0)
        {
            n /= 2;
        }
    }
    // 2. Xu ly cac SNT le p >= 3
    long long p = 3;
    while (p * p <= n)
    {
        if (n % p == 0)
        {
            // Ap dung cong thuc: result = result - result / p
            result = result - result / p;
            // Loai bo het thua so p
            while (n % p == 0)
            {
                n /= p;
            }
        }
        // TANG p DUNG VI TRI: Sau khi xu ly xong TAT CA luy thua cua p,
        // ta moi chuyen sang so le tiep theo.
        p += 2;
    }
    // 3. Neu n con lai la SNT lon (> sqrt ban dau)
    if (n > 1)
    {
        result = result - result / n;
    }
    return result;
}
int main()
{
    cout << "Phi(10) = " << euler_phi(10) << endl; // 4
    cout << "Phi(17) = " << euler_phi(17) << endl; // 16
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
Phi(10) = 4
Phi(17) = 16
```

- Giải thích Code C++:

Kiểu dữ liệu: Bắt buộc dùng **long long** cho **n** và **result** vì **n** có thể lên đến 10^{12} hoặc 10^{18}. Việc tách p = 2 và dùng p+=2 giúp tối ưu code C++.

V. Lý Thuyết: Định Lý Euler (10 phút)

1. Định lý Fermat nhỏ (Fermat's Little Theorem)

- Đây là trường hợp đặc biệt của Định lý Euler: Nếu p là số nguyên tố, và a không chia hết cho p, thì:

$$a^{p-1} \equiv 1 \pmod{p}$$

- **Lưu ý:**

$$p - 1 = \phi(p) \text{ (vì } \phi(\text{số nguyên tố } p) = p - 1).$$

2. Định Lý Euler (Euler's Theorem)

- Định lý tổng quát hơn Định lý Fermat nhỏ: Nếu ƯCLN(a, n) = 1 (a và n nguyên tố cùng nhau), thì:

$$a^{\phi(n)} \equiv 1 \pmod{n}$$

- **Ý nghĩa quan trọng:** Khi tính A^B (mod M), nếu ƯCLN(A, M) = 1, ta có thể thay thế số mũ B bằng B (mod ∅(M)).

3. Định lý Euler Mở Rộng (Euler's Totient Theorem Extension)

- Điều gì xảy ra nếu ƯCLN(a, n) ≠ 1? Ta vẫn có công thức giảm mũ, nhưng có một sự thay đổi nhỏ:

$$a^B \equiv a^{B \pmod{\phi(n)} + \phi(n)} \pmod{n} \quad \text{khi } B \geq \phi(n)$$

- **Tóm lại:** Khi B rất lớn, ta luôn giảm số mũ B xuống **B' = B (mod ∅(n))**

  + Nếu B' quá nhỏ (ví dụ B' < ∅(n)) và phép chia B cho ∅(n) có dư, ta dùng B'.

  + Nếu B' = 0 hoặc để đảm bảo tính an toàn cho mọi trường hợp (B đủ lớn), ta dùng **B' = B (mod ∅(n)) + ∅(n)**

VI. Thực Hành Kèm Code Mẫu:

1. Bài 2. Ứng dụng Định lý Euler (A^B (mod M) với B rất lớn) (20 phút)

- **Yêu cầu:** Tính **A^B (mod M)**, với A, M nhỏ và B là một số rất lớn (có thể lên tới 10^{1000}).

- **Logic:**

  + Tính **∅(M)** bằng hàm **euler_phi** đã viết.

  + Tính số mũ mới B' = B (mod ∅(M)) (hoặc B (mod ∅(M)) + ∅(M), nếu B ≥ ∅(M))

  + Dùng thuật toán Lũy thừa nhanh (đã học ở Buổi 10) để tính:

  $A^{B'}$ (mod M).

- **Lưu ý:** Vì B quá lớn, ta không thể lưu B vào biến long long. Ta phải đọc B dưới dạng chuỗi và tính B (mod ∅(M)) trực tiếp từ chuỗi đó.

1.1. Code Python:

```python
import math
# --- HÀM LŨY THỪA NHANH (POWER) ---
def power(base, exponent, modulus):
    # Tính (base^exponent) mod modulus.
    result = 1
    base = base % modulus
    while exponent > 0:
        if exponent % 2 == 1:
            result = (result * base) % modulus
        base = (base * base) % modulus
        exponent = exponent // 2
    return result
# --- HÀM EULER_PHI ---
def euler_phi(n):
    if n == 0:
        return 0
```

```
    result = n
    # 1. Xử lý SNT p=2
    if n % 2 == 0:
        result = result - result // 2
        while n % 2 == 0:
            n //= 2
    # 2. Xử lý các SNT lẻ p >= 3
    p = 3
    while p * p <= n:
        if n % p == 0:
            result = result - result // p
            while n % p == 0:
                n //= p
        p += 2
    # 3. Neu n con lai la SNT lon (> sqrt ban dau)
    if n > 1:
        result = result - result // n
    return result
# --- HÀM TÍNH B MOD PHI(M) (B LÀ CHUỖI) ---
def get_exponent_mod_phi(B_str, phi_M):
    mod_val = 0
    for digit in B_str:
        mod_val = (mod_val * 10 + int(digit)) % phi_M
    return mod_val
# --- HÀM CHÍNH (ĐỊNH LÝ EULER MỞ RỘNG) ---
def big_power_mod_M(A, B_str, M):
    phi_M = euler_phi(M)
    # 1. Tinh B mod phi(M)
    exponent_reduced = get_exponent_mod_phi(B_str, phi_M)
    # Kiem tra xem B co lon hon hoac bang phi_M khong
    is_B_ge_phi = len(B_str) > len(str(phi_M)) or \
                  (len(B_str) == len(str(phi_M)) and B_str >= str(phi_M))
    # 2. ÁP DỤNG ĐỊNH LÝ EULER MỞ RỘNG (Điều chỉnh số mũ)
    # Nếu B >= phi(M), ta phải đảm bảo số mũ mới >= phi(M).
    # Công thức an toàn là B' = (B mod phi(M)) + phi(M)
    # Tuy nhiên, nếu B mod phi(M) = 0 và B >= phi(M), ta dùng B' = phi(M).
    if is_B_ge_phi:
        # Nếu B >= phi(M), ta luôn cộng phi(M) vào (B mod phi(M)).
        # Kể cả khi (B mod phi(M)) = 0, nó trở thành phi(M).
        exponent_reduced += phi_M
    # 3. Tinh A^B' mod M bang Luy thua nhanh (Lệnh RETURN PHẢI Ở CUỐI)
    return power(A, exponent_reduced, M)
# ----------------- CHẠY THỬ -----------------
B_string = "100"
# Kiem tra A=3, B=100 (chuoi), M=7. Phi(7)=6.
# B mod Phi(7) = 100 mod 6 = 4.
# B > Phi(7) => exponent_reduced = 4 + 6 = 10.
# Ket qua can tim: 3^10 mod 7 = (3^6 * 3^4) mod 7 = (1 * 3^4) mod 7 = 81 mod 7 = 4.
print(f"3^{B_string} mod 7: {big_power_mod_M(3, B_string, 7)}") # 4
# Ví dụ 2: B = "6" (B = phi(M))
# B mod Phi(7) = 6 mod 6 = 0. is_B_ge_phi = True. exponent_reduced = 0 + 6 = 6.
# Kết quả: 3^6 mod 7 = 1
B_string_2 = "6"
print(f"3^{B_string_2} mod 7: {big_power_mod_M(3, B_string_2, 7)}") # 1
```

=> Kết quả sau khi chạy chương trình:

```
3^100 mod 7: 4
3^6 mod 7: 1
```

- Giải thích Code Python:

+ **Xử lý chuỗi:** Hàm **get_exponent_mod_phi** là bắt buộc để tính **B (mod ∅(M))** khi B là chuỗi dài.

+ **Công thức mở rộng:** Việc kiểm tra **B ≥ ∅(M)** và sau đó thêm ∅(M) vào **exponent_reduced** là cần thiết để áp dụng Định lý Euler mở rộng một cách an toàn cho mọi trường hợp.

1.2. Code C++:

```
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
#include <string>            // Thư viện xử lí chuỗi
#include <cmath>             // Thư viện toán học
using namespace std;
// --- HAM LUY THUA NHANH (POWER) ---
long long power(long long base, long long exponent, long long modulus)
{
    // Tinh (base^exponent) mod modulus.
    long long result = 1;
    base = base % modulus;
    while (exponent > 0)
    {
        if (exponent % 2 == 1)
        {
            result = (result * base) % modulus;
        }
        base = (base * base) % modulus;
        exponent = exponent / 2;
    }
    return result;
}
// --- HAM EULER_PHI ---
long long euler_phi(long long n)
{
    if (n == 0)
    {
        return 0;
    }
    long long result = n;
    // 1. Xu ly SNT p=2
    if (n % 2 == 0)
    {
        result = result - result / 2;
        while (n % 2 == 0)
        {
            n /= 2;
        }
    }
    // 2. Xu ly cac SNT le p >= 3
    long long p = 3;
    while (p * p <= n)
    {
        if (n % p == 0)
        {
            result = result - result / p;
            while (n % p == 0)
            {
                n /= p;
```

```
            }
        }
        p += 2;
    }
    // 3. Neu n con lai la SNT lon (> sqrt ban dau)
    if (n > 1)
    {
        result = result - result / n;
    }
    return result;
}
// --- HAM TINH B MOD PHI(M) (B LA CHUOI) ---
long long get_exponent_mod_phi(const string &B_str, long long phi_M)
{
    long long mod_val = 0;
    for (char digit : B_str)
    {
        mod_val = (mod_val * 10 + (digit - '0')) % phi_M;
    }
    return mod_val;
}
// --- HAM CHINH (DINH LY EULER MO RONG) ---
long long big_power_mod_M(long long A, const string &B_str, long long M)
{
    long long phi_M = euler_phi(M);
    // 1. Tinh B mod phi(M)
    long long exponent_reduced = get_exponent_mod_phi(B_str, phi_M);
    // Kiem tra xem B co lon hon hoac bang phi_M khong
    bool is_B_ge_phi =
        (B_str.size() > to_string(phi_M).size()) ||
        (B_str.size() == to_string(phi_M).size() && B_str >= to_string(phi_M));
    // 2. AP DUNG DINH LY EULER MO RONG (Dieu chinh so mu)
    // Neu B >= phi(M), ta phai dam bao so mu moi >= phi(M).
    // Cong thuc an toan la B' = (B mod phi(M)) + phi(M)
    // Tuy nhien, neu B mod phi(M) = 0 va B >= phi(M), ta dung B' = phi(M).
    if (is_B_ge_phi)
    {
        // Neu B >= phi(M), ta luon cong phi(M) vao (B mod phi(M)).
        // Ke ca khi (B mod phi(M)) = 0, no tro thanh phi(M).
        exponent_reduced += phi_M;
    }
    // 3. Tinh A^B' mod M bang Luy thua nhanh (Lenh RETURN PHAI O CUOI)
    return power(A, exponent_reduced, M);
}
// ----------------- CHAY THU -----------------
int main()
{
    string B_string = "100";
    // Kiem tra A=3, B=100 (chuoi), M=7. Phi(7)=6.
    // B mod Phi(7) = 100 mod 6 = 4.
    // B > Phi(7) => exponent_reduced = 4 + 6 = 10.
    // Ket qua can tim: 3^10 mod 7 = (3^6 * 3^4) mod 7 = (1 * 3^4) mod 7 = 81 mod 7
= 4.
     cout << "3^" << B_string << " mod 7: " << big_power_mod_M(3, B_string, 7) <<
endl;
    // Vi du 2: B = "6" (B = phi(M))
    // B mod Phi(7) = 6 mod 6 = 0. is_B_ge_phi = True. exponent_reduced = 0 + 6 = 6.
    // Ket qua: 3^6 mod 7 = 1
```

```
    string B_string_2 = "6";
    cout << "3^" << B_string_2 << " mod 7: " << big_power_mod_M(3, B_string_2, 7) << endl;
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
3^100 mod 7: 4
3^6 mod 7: 1
```

- Giải thích Code C++:

+ **An toàn:** Việc kiểm tra **is_B_ge_phi** được thực hiện cẩn thận bằng cách so sánh độ dài chuỗi và chuỗi để đảm bảo **exponent_reduced** được cộng **ø(M)** đúng lúc.

+ **Lưu ý:** Phải đảm bảo **power** và **euler_phi** sử dụng long long.

VII. Bài Tập Về Nhà Kèm Gợi Ý Code:

1. Bài 1. Tính tổng $\sum_{i=1}^{N} \phi(i)$

- **Mô tả:** Tính tổng giá trị của Hàm Phi Euler từ 1 đến N.
- **Logic:** Sử dụng Sàng Eratosthenes để tính **ø(i)** cho tất cả i từ 1 đến N trong thời gian O(N log log N).
- **Gợi ý:** Khởi tạo **phi[i] = i**. Sau đó, khi gặp **SNTp**, cập nhật:

  phi[i] = phi[i] . (1 - 1/p) cho tất cả bội số i của p.

1.1. Code Python:

```python
def sum_of_euler_phi(N):
    # Tính tổng Sigma(phi(i)) từ i=1 đến N bằng Sàng Eratosthenes.
    if N <= 0:
        return 0
    # 1. Khởi tạo mảng phi. phi[i] = i
    # Kích thước N + 1 để sử dụng chỉ số từ 0 đến N
    phi = list(range(N + 1))
    # 2. Áp dụng Sàng Eratosthenes (Duyệt từ 2 đến N)
    for i in range(2, N + 1):
        if phi[i] == i:
            # i là số nguyên tố (SNT p)
            # Cập nhật phi[p]: phi(p) = p - 1
            phi[i] -= 1
            # Cập nhật cho các bội số j = i * k
            for j in range(2 * i, N + 1, i):
                # Áp dụng công thức: phi[j] = phi[j] - phi[j] // i
                # (Vì i là ước nguyên tố của j)
                phi[j] -= phi[j] // i
    # 3. Tính tổng Sigma(phi(i))
    total_sum = 0
    # Bắt đầu từ i=1 vì phi(1) = 1 (đã được khởi tạo đúng)
    for i in range(1, N + 1):
        total_sum += phi[i]
    return total_sum
# ----------------- CHẠY THỬ -----------------
N1 = 10
# Phi(1)=1, Phi(2)=1, Phi(3)=2, Phi(4)=2, Phi(5)=4,
# Phi(6)=2, Phi(7)=6, Phi(8)=4, Phi(9)=6, Phi(10)=4
# Tổng = 1+1+2+2+4+2+6+4+6+4 = 32
print(f"Tổng Phi(i) từ 1 đến {N1}: {sum_of_euler_phi(N1)}")
N2 = 20
print(f"Tổng Phi(i) từ 1 đến {N2}: {sum_of_euler_phi(N2)}") # Kết quả: 124
```

=> Kết quả sau khi chạy chương trình:

```
Tổng Phi(i) từ 1 đến 10: 32
Tổng Phi(i) từ 1 đến 20: 128
```

1.2. Code C++:

```
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
#include <string>            // Thư viện xử lí chuỗi
#include <cmath>             // Thư viện toán học
using namespace std;
long long sum_of_euler_phi(int N)
{
    // Tinh tong Sigma(phi(i)) tu i=1 den N bang Sang Eratosthenes.
    if (N <= 0)
    {
        return 0;
    }
    // 1. Khoi tao mang phi. phi[i] = i
    // Kich thuoc N + 1 de su dung chi so tu 0 den N
    vector<long long> phi(N + 1);
    for (int i = 0; i <= N; i++)
    {
        phi[i] = i;
    }
    // 2. Ap dung Sang Eratosthenes (Duyet tu 2 den N)
    for (int i = 2; i <= N; i++)
    {
        if (phi[i] == i)
        {
            // i la so nguyen to (SNT p)
            // Cap nhat phi[p]: phi(p) = p - 1
            phi[i] -= 1;
            // Cap nhat cho cac boi so j = i * k
            for (int j = 2 * i; j <= N; j += i)
            {
                // Ap dung cong thuc: phi[j] = phi[j] - phi[j] // i
                // (Vi i la uoc nguyen to cua j)
                phi[j] -= phi[j] / i;
            }
        }
    }
    // 3. Tinh tong Sigma(phi(i))
    long long total_sum = 0;
    // Bat dau tu i=1 vi phi(1) = 1 (da duoc khoi tao dung)
    for (int i = 1; i <= N; i++)
    {
        total_sum += phi[i];
    }
    return total_sum;
}
int main()
{
    // ----------------- CHAY THU -----------------
    int N1 = 10;
    // Phi(1)=1, Phi(2)=1, Phi(3)=2, Phi(4)=2, Phi(5)=4,
    // Phi(6)=2, Phi(7)=6, Phi(8)=4, Phi(9)=6, Phi(10)=4
    // Tong = 1+1+2+2+4+2+6+4+6+4 = 32
    cout << "Tong Phi(i) tu 1 den " << N1 << ": " << sum_of_euler_phi(N1) << "\n";
```

```
    int N2 = 20;
    cout << "Tong Phi(i) tu 1 den " << N2 << ": " << sum_of_euler_phi(N2) << "\n";
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
Tổng Phi(i) từ 1 đến 10: 32
Tổng Phi(i) từ 1 đến 20: 128
```

2. Bài 2. Ứng dụng: Chu kỳ của a^n (mod M).

- **Mô tả:** Tìm chu kỳ của dãy a^1, a^2, a^3, … (mod M).
- **Logic:** Chu kỳ là ước của **ø(M)**. Ta chỉ cần tìm k nhỏ nhất sao cho:

$$a^k \equiv 1 \pmod{M}$$

(với ƯCLN(a, M) = 1).

- **Gợi ý:**
  + Tính **ø(M)**.
  + Phân tích **ø(M)** thành các thừa số nguyên tố.
  + Duyệt qua tất cả các ước của **ø(M)** theo thứ tự tăng dần, dùng **power** để kiểm tra điều kiện đồng dư.

2.1. Code Python:

```
import math
# --- 1. HÀM LŨY THỪA NHANH (power) ---
def power(base, exponent, modulus):
    # Tính (base^exponent) mod modulus.
    result = 1
    base = base % modulus
    while exponent > 0:
        if exponent % 2 == 1:
            result = (result * base) % modulus
        base = (base * base) % modulus
        exponent = exponent // 2
    return result
# --- 2. HÀM EULER PHI (euler_phi) ---
def euler_phi(n):
    # Tính Hàm Phi Euler phi(n) O(sqrt(n)).
    if n <= 0: return 0
    result = n
    # Xử lý SNT p=2
    if n % 2 == 0:
        result -= result // 2
        while n % 2 == 0: n //= 2
    # Xử lý các SNT lẻ p >= 3
    p = 3
    while p * p <= n:
        if n % p == 0:
            result -= result // p
            while n % p == 0: n //= p
        p += 2
    # Phần còn lại (nếu n là SNT lớn)
    if n > 1: result -= result // n
    return result
# --- 3. HÀM CHÍNH: TÌM CHU KỲ (find_order) ---
def find_order(a, M):
    # Tìm k nhỏ nhất (cấp của a mod M) sao cho a^k = 1 (mod M).
    # Yêu cầu: math.gcd(a, M) == 1.
```

```
    if math.gcd(a, M) != 1:
        # Trường hợp không nguyên tố cùng nhau thì không áp dụng Định lý Euler.
        return "Loi: a va M khong nguyen to cung nhau."
    phi_M = euler_phi(M)
    # Phân tích phi_M thành thừa số nguyên tố duy nhất
    prime_factors = set()
    temp_phi = phi_M
    d = 2
    while d * d <= temp_phi:
        if temp_phi % d == 0:
            prime_factors.add(d)
            while temp_phi % d == 0:
                temp_phi //= d
        d += 1
    if temp_phi > 1:
        prime_factors.add(temp_phi)
    # Chu kỳ k khởi tạo là phi_M
    min_order = phi_M
    # Duyệt qua các thừa số nguyên tố p của phi_M và thử giảm k
    for p in prime_factors:
        # Thử chia min_order cho p (lặp lại cho tất cả lũy thừa của p)
        while min_order % p == 0:
            new_order = min_order // p
            # Kiểm tra a^(k/p) mod M = 1?
            if power(a, new_order, M) == 1:
                min_order = new_order # Cấp nhỏ hơn được tìm thấy
            else:
                break # Không thể chia cho p nữa
    return min_order
# ----------------- CHẠY THỬ -----------------
# Ví dụ 1: a=3, M=7. Phi(7)=6.
# Chu kỳ phải là 6 (3^6 = 729 = 1 mod 7).
A1 = 3
M1 = 7
print(f"Chu kỳ của {A1}^n mod {M1}: {find_order(A1, M1)}") # Kết quả: 6
# Ví dụ 2: a=2, M=15. Phi(15) = 15 * (1-1/3) * (1-1/5) = 8. Các ước: 1, 2, 4, 8.
# 2^4 = 16 = 1 mod 15. Chu kỳ là 4.
A2 = 2
M2 = 15
print(f"Chu kỳ của {A2}^n mod {M2}: {find_order(A2, M2)}") # Kết quả: 4
```

=> Kết quả sau khi chạy chương trình:

```
Chu kỳ của 3^n mod 7: 6
Chu kỳ của 2^n mod 15: 4
```

2.2. Code C++:

```
#include <iostream>         // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>    // Thư viện tổng hợp
#include <string>           // Thư viện xử lí chuỗi
#include <cmath>            // Thư viện toán học
using namespace std;
// --- 1. HAM LUY THUA NHANH (power) ---
long long power(long long base, long long exponent, long long modulus)
{
    // Tinh (base^exponent) mod modulus.
    long long result = 1;
    base = base % modulus;
    while (exponent > 0)
    {
```

```
        if (exponent % 2 == 1)
        {
            result = (result * base) % modulus;
        }
        base = (base * base) % modulus;
        exponent = exponent / 2;
    }
    return result;
}
// --- 2. HAM EULER PHI (euler_phi) ---
long long euler_phi(long long n)
{
    // Tinh Ham Phi Euler phi(n) O(sqrt(n)).
    if (n <= 0) return 0;
    long long result = n;
    // Xu ly SNT p=2
    if (n % 2 == 0)
    {
        result -= result / 2;
        while (n % 2 == 0) n /= 2;
    }
    // Xu ly cac SNT le p >= 3
    long long p = 3;
    while (p * p <= n)
    {
        if (n % p == 0)
        {
            result -= result / p;
            while (n % p == 0) n /= p;
        }
        p += 2;
    }
    // Phan con lai (neu n la SNT lon)
    if (n > 1) result -= result / n;
    return result;
}
// --- 3. HAM CHINH: TIM CHU KY (find_order) ---
long long find_order(long long a, long long M)
{
    // Tim k nho nhat (cap cua a mod M) sao cho a^k = 1 (mod M).
    // Yeu cau: gcd(a, M) == 1.
    if (std::gcd(a, M) != 1)
    {
        // Truong hop khong nguyen to cung nhau thi khong ap dung Dinh ly Euler.
        cout << "Loi: a va M khong nguyen to cung nhau.";
        return -1;
    }
    long long phi_M = euler_phi(M);
    // Phan tich phi_M thanh thua so nguyen to duy nhat
    set<long long> prime_factors;
    long long temp_phi = phi_M;
    long long d = 2;
    while (d * d <= temp_phi)
    {
        if (temp_phi % d == 0)
        {
            prime_factors.insert(d);
            while (temp_phi % d == 0)
```

```
            {
                temp_phi /= d;
            }
        }
        d++;
    }
    if (temp_phi > 1)
    {
        prime_factors.insert(temp_phi);
    }
    // Chu ky k khoi tao la phi_M
    long long min_order = phi_M;
    // Duyet qua cac thua so nguyen to p cua phi_M va thu giam k
    for (long long p : prime_factors)
    {
        // Thu chia min_order cho p (lap lai cho tat ca luy thua cua p)
        while (min_order % p == 0)
        {
            long long new_order = min_order / p;
            // Kiem tra a^(k/p) mod M = 1?
            if (power(a, new_order, M) == 1)
            {
                min_order = new_order; // Cap nho hon duoc tim thay
            }
            else
            {
                break; // Khong the chia cho p nua
            }
        }
    }
    return min_order;
}
// ----------------- CHAY THU -----------------
int main()
{
    // Vi du 1: a=3, M=7. Phi(7)=6.
    // Chu ky phai la 6 (3^6 = 729 = 1 mod 7).
    long long A1 = 3, M1 = 7;
    cout << "Chu ky cua " << A1 << "^n mod " << M1 << ": " << find_order(A1, M1) <<
"\n";
    // Vi du 2: a=2, M=15. Phi(15)=8. Chu ky la 4.
    long long A2 = 2, M2 = 15;
    cout << "Chu ky cua " << A2 << "^n mod " << M2 << ": " << find_order(A2, M2) <<
"\n";
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
Chu kỳ của 3^n mod 7: 6
Chu kỳ của 2^n mod 15: 4
```

VIII. Tổng Kết & Dặn Dò:

1. Củng cố chính:

- **∅(n)** đếm số lượng số nguyên tố cùng nhau với n và ≤ n.
- **∅(n)** được tính hiệu quả bằng phân tích thừa số nguyên tố O(sqrt(n)).
- **Định lý Euler** cho phép giảm số mũ cực lớn **B** xuống **B (mod ∅(M))**

hoặc: $\mathbf{B} \pmod{\phi(\mathbf{M})} + \phi(\mathbf{M}))$ khi tính $\mathbf{A}^{\mathbf{B}} \pmod{\mathbf{M}}$

2. Dặn dò: Hoàn thành bài tập về nhà. Buổi tiếp theo chúng ta sẽ đi vào chủ đề **Hình học: Khoảng cách, diện tích, kiểm tra thẳng hàng.**

BÀI 12. HÌNH HỌC CƠ BẢN
KHOẢNG CÁCH, DIỆN TÍCH, KIỂM TRA THẲNG HÀNG

I. Mục Tiêu:

- **Kiến thức:** Học sinh (**HS**) hiểu cách biểu diễn điểm và vector trong lập trình. Nắm vững công thức tính khoảng cách, diện tích tam giác và kiểm tra vị trí tương đối (thẳng hàng, rẽ trái/phải).
- **Kỹ năng:** HS thành thạo việc cài đặt các hàm hình học cơ bản, đặc biệt là sử dụng **Tích có hướng 2D (Cross Product)** để giải quyết các bài toán về diện tích và định hướng.
- **Tư duy:** HS áp dụng kiến thức số học (kiểu dữ liệu float/double và so sánh xấp xỉ) vào bài toán hình học.

II. Phân Bổ Thời Gian (90 Phút):

Hoạt Động	Nội Dung	Thời gian (Phút)
Khởi động	Ôn tập Tọa độ, giới thiệu Vector	10
Lý thuyết 1	Cấu trúc Điểm, Khoảng cách Euclid	15
Thực hành 1	Bài 1. Tính Khoảng cách & Chu vi	15
Lý thuyết 2	Tích có hướng 2D (Cross Product)	15
Thực hành 2	Bài 2. Diện tích Tam giác, Kiểm tra Thẳng hàng/Định hướng	30
Tổng kết	Bài tập về nhà, Củng cố	5
Tổng cộng		**90**

III. Lý Thuyết 1: Điểm và Khoảng cách (15 phút)

1. Biểu diễn Điểm (Point):

Trong lập trình, điểm (x, y) thường được biểu diễn bằng một cấu trúc (struct trong C++) hoặc một lớp/tuple trong Python.

Ngôn Ngữ	Cấu Trúc	Giải Thích
Python	point = (x, y)	Thường dùng tuple hoặc namedtuple.
C++	struct Point {double x, y;};	Sử dụng struct với kiểu double để đảm bảo độ chính xác.

2. Khoảng cách Euclid (Euclidean Distance)

- Khoảng cách giữa hai điểm **A(x_1, y_2)** và **B(x_2, y_2)**:

$$d(A, B) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

- Công thức tính khoảng cách từ một điểm **P(x_1, y_1, z_1)** đến mặt phẳng **Ax + By + Cz + D = 0:**

$$\text{distance} = \frac{|Ax_1 + By_1 + Cz_1 + D|}{\sqrt{A^2 + B^2 + C^2}}$$

IV. Thực Hành 1: Tính Khoảng cách và Chu vi (15 phút)

1. Bài 1. Hàm tính Khoảng cách

- **Yêu cầu:** Viết hàm tính khoảng cách Euclid giữa hai điểm A và B.

1.1. Code Python:

```python
import math # Thu vien toan hoc, de tinh can bac 2
# Ham tinh khoang cach giua 2 diem A va B
# A = (x1, y1), B = (x2, y2)
def distance(A, B):
    # A = (x1, y1), B = (x2, y2)
    dx = A[0] - B[0]
    dy = A[1] - B[1]
    return math.sqrt(dx*dx + dy*dy)
# Khai bao A = (1.0, 2.0), B = (4.0, 6.0)
A = (1.0, 2.0)
B = (4.0, 6.0)
print(f"Khoang cach: {distance(A, B)}") # 5.0
```

=> Kết quả sau khi chạy chương trình:

```
Khoang cach: 5.0
```

- Giải thích Code Python:

Kiểu dữ liệu: Python tự động xử lý số thực. math.sqrt được sử dụng để tính căn bậc hai.

1.2. Code C++:

```cpp
#include <iostream>         // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>    // Thư viện tổng hợp
#include <cmath>            // Thư viện toán học, để tính căn bậc 2
using namespace std;
// Ham tinh khoang cach giua 2 diem A va B
// A = (x1, y1), B = (x2, y2)
double distanceAB(double x1, double y1, double x2, double y2)
{
    double dx = x1 - x2;
    double dy = y1 - y2;
    return sqrt(dx*dx + dy*dy);
}
int main()
{
    // Khai bao A = (1.0, 2.0), B = (4.0, 6.0)
    double x1 = 1.0, y1 = 2.0;
    double x2 = 4.0, y2 = 6.0;
    cout << "Khoang cach: " << distanceAB(x1, y1, x2, y2);  // 5.0
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
Khoang cach: 5.0
```

- Giải thích Code C++:

Thư viện: Cần thư viện **cmath** cho hàm **std::sqrt**. Quan trọng là sử dụng **double** cho tọa độ và kết quả.

V. Lý Thuyết 2: Tích có hướng 2D (20 phút)

1. Tích có hướng 2D (Cross Product/Determinant)

Đây là công cụ quan trọng nhất trong hình học phẳng. Đối với hai vector **a** = **(x_a, y_a)** và **b** = **(x_b, y_b)**, tích có hướng 2D (được định nghĩa là thành phần **z** của tích có hướng 3D) là: **Cross(a, b) = $x_a y_b - x_b y_a$**

2. Ứng dụng quan trọng nhất (Với 3 điểm A, B, C)

- Ta tính tích có hướng của hai vector **AB = ($x_B - x_A$, $y_B - y_A$)** và **BC = ($x_C - x_B$, $y_C - y_B$)**.

- **Diện tích Tam giác ABC:**

$$\text{Area}(\mathbf{ABC}) = \frac{1}{2} \cdot |\text{Cross}(\mathbf{AB}, \mathbf{AC})|$$

- **Kiểm tra Định hướng (Orientation)**: Dấu của Cross(AB, AC) cho biết vị trí của C so với đoạn thẳng AB.

  + **Kết quả = 0**: Ba điểm A, B, C **thẳng hàng**.
  + **Kết quả > 0**: C nằm bên **trái** AB (xoay ngược chiều kim đồng hồ).
  + **Kết quả < 0**: C nằm bên **phải** AB (xoay thuận chiều kim đồng hồ).

VI. Thực Hành 2: Diện tích, Thẳng hàng, Định hướng (30 phút)

1. Bài 2. Hàm Tích có hướng, Diện tích và Kiểm tra

- **Yêu cầu:** Viết hàm cross_product(A, B, C) để tính Cross(AB, AC), sau đó áp dụng cho diện tích và kiểm tra thẳng hàng.

1.1. Code Python:

```
import math # Thu vien toan hoc, de tinh can bac 2
# Hằng số sai số nhỏ (Epsilon)
# Dùng để so sánh số thực với 0.
EPSILON = 1e-9
# Giả định điểm được truyền vào dưới dạng tuple: (x, y)
# Ví dụ: A = (0.0, 0.0)
def cross_product(A, B, C):
    # Tính tích có hướng 2D của vector AB và vector AC.
    # Kết quả: (Bx - Ax) * (Cy - Ay) - (By - Ay) * (Cx - Ax)
    # Vector AB: (B[0] - A[0], B[1] - A[1])
    # Vector AC: (C[0] - A[0], C[1] - A[1])
    cross = (B[0] - A[0]) * (C[1] - A[1]) - (B[1] - A[1]) * (C[0] -
A[0])
    return cross
def area_triangle(A, B, C):
    # Tính diện tích tam giác ABC.
    # Diện tích = 0.5 * |Tích có hướng(AB, AC)|
    cp = cross_product(A, B, C)
    return 0.5 * abs(cp)
def is_collinear(A, B, C):
    # Kiểm tra ba điểm A, B, C có thẳng hàng hay không.
    # Thẳng hàng khi Tích có hướng bằng 0 (trong phạm vi sai số EPSILON).
    cp = cross_product(A, B, C)
    # Kiểm tra liệu giá trị tuyệt đối của cp có nhỏ hơn EPSILON không
    return abs(cp) < EPSILON
# ----------------- CHẠY THỬ -----------------
# 1. Ví dụ tính Diện tích
A1 = (0.0, 0.0)
B1 = (4.0, 0.0)
```

```
C1 = (0.0, 3.0) # Tam giác vuông, Diện tích = 0.5 * 4 * 3 = 6.0
print(f"Diện tích tam giác A1B1C1: {area_triangle(A1, B1, C1)}")    # 6.0
# 2. Ví dụ kiểm tra Thẳng hàng
A2 = (0.0, 0.0)
B2 = (1.0, 1.0)
C2 = (2.0, 2.0) # Thẳng hàng
print(f"A2, B2, C2 thẳng hàng: {is_collinear(A2, B2, C2)}")         # True
# 3. Ví dụ không Thẳng hàng
A3 = (0.0, 0.0)
B3 = (1.0, 1.0)
C3 = (1.0, 2.0)
print(f"A3, B3, C3 thẳng hàng: {is_collinear(A3, B3, C3)}")         # False
```

=> Kết quả sau khi chạy chương trình:

```
Dien tich tam giac A1B1C1: 6.0
A2, B2, C2 thang hang: True
A3, B3, C3 thang hang: False
```

- Giải thích Code Python: Tích có hướng

1.2. Code C++:

```
#include <iostream>         // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>    // Thư viện tổng hợp
#include <cmath>            // Thư viện toán học, để tính căn bậc 2
using namespace std;
// Hang so sai so nho (Epsilon)
// Dung de so sanh so thuc voi 0.
const double EPSILON = 1e-9;
// Gia dinh diem duoc truyen vao duoi dang struct: (x, y)
// Vi du: A = (0.0, 0.0)
struct Point
{
    double x, y;
};
// Ham tinh tich co huong 2D cua vector AB va vector AC.
// Ket qua: (Bx - Ax) * (Cy - Ay) - (By - Ay) * (Cx - Ax)
// Vector AB: (B.x - A.x, B.y - A.y)
// Vector AC: (C.x - A.x, C.y - A.y)
double cross_product(Point A, Point B, Point C)
{
    double cross = (B.x - A.x) * (C.y - A.y) - (B.y - A.y) * (C.x - A.x);
    return cross;
}
// Tinh dien tich tam giac ABC.
// Dien tich = 0.5 * |Tich co huong(AB, AC)|
double area_triangle(Point A, Point B, Point C)
{
    double cp = cross_product(A, B, C);
    return 0.5 * fabs(cp);
}
// Kiem tra ba diem A, B, C co thang hang hay khong.
// Thang hang khi Tich co huong bang 0 (trong pham vi sai so EPSILON).
bool is_collinear(Point A, Point B, Point C)
{
    double cp = cross_product(A, B, C);
    // Kiem tra lieu gia tri tuyet doi cua cp co nho hon EPSILON khong
    return fabs(cp) < EPSILON;
}
// ----------------- CHAY THU -----------------
```

```
int main()
{
    // 1. Vi du tinh Dien tich
    Point A1 = {0.0, 0.0};
    Point B1 = {4.0, 0.0};
    Point C1 = {0.0, 3.0}; // Tam giac vuong, Dien tich = 0.5 * 4 * 3 = 6.0
    cout << "Dien tich tam giac A1B1C1: " << area_triangle(A1, B1, C1) << endl; //
6.0
    // 2. Vi du kiem tra Thang hang
    Point A2 = {0.0, 0.0};
    Point B2 = {1.0, 1.0};
    Point C2 = {2.0, 2.0}; // Thang hang
    cout << "A2, B2, C2 thang hang: " << (is_collinear(A2, B2, C2) ? "True" :
"False") << endl; // True
    // 3. Vi du khong Thang hang
    Point A3 = {0.0, 0.0};
    Point B3 = {1.0, 1.0};
    Point C3 = {1.0, 2.0};
    cout << "A3, B3, C3 thang hang: " << (is_collinear(A3, B3, C3) ? "True" :
"False") << endl; // False
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
Dien tich tam giac A1B1C1: 6.0
A2, B2, C2 thang hang: True
A3, B3, C3 thang hang: False
```

- Giải thích Code C++:

So sánh với 0: Sử dụng **const double EPS = 1e-9**; (hoặc nhỏ hơn) là **BẮT BUỘC** khi làm việc với số thực trong C++.

VII. Bài Tập Về Nhà & Tổng Kết (5 phút)

1. Bài tập về nhà:

1.1. Bài tập 1 (Kiểm tra hình): Cho 4 điểm A, B, C, D. Kiểm tra xem A, B, C, D có tạo thành hình bình hành hay không (Kiểm tra bằng cách so sánh độ dài các cạnh đối và/hoặc kiểm tra vector bằng nhau).

1.1.1. Code Python:

```
import math # Thư viện toán học
def get_vector(P, Q):
    # Tính vector PQ = (xQ - xP, yQ - yP)
    # P = (x1, y1), Q = (x2, y2)
    return (Q[0] - P[0], Q[1] - P[1])
def is_parallelogram(A, B, C, D):
    # Kiểm tra xem 4 điểm A, B, C, D có tạo thành hình bình hành hay không
    # theo thứ tự A -> B -> C -> D.
    # Logic: Kiểm tra xem các vector đối có bằng nhau không:
    #- Vector AB có bằng Vector DC không? (AB = DC)
    #- Vector AD có bằng Vector BC không? (AD = BC)
    # Lưu ý: Nếu dùng tọa độ số thực (float), nên sử dụng sai số EPSILON
    # để so sánh, nhưng với bài toán cơ bản này, ta giả định so sánh trực tiếp
    # các vector là đủ.
    # 1. Tính các vector cần thiết
    vector_AB = get_vector(A, B)    # (x_B - x_A, y_B - y_A)
    vector_DC = get_vector(D, C)    # (x_C - x_D, y_C - y_D)
    vector_AD = get_vector(A, D)    # (x_D - x_A, y_D - y_A)
    vector_BC = get_vector(B, C)    # (x_C - x_B, y_C - y_B)
```

```
    # 2. Kiểm tra điều kiện hình bình hành
    # Điều kiện 1: Vector AB phải bằng Vector DC (AB || DC và |AB| = |DC|)
    condition1 = (vector_AB == vector_DC)
    # Điều kiện 2: Vector AD phải bằng Vector BC (AD || BC và |AD| = |BC|)
    condition2 = (vector_AD == vector_BC)
    # Hình bình hành khi cả hai điều kiện thỏa mãn
    return condition1 and condition2
# ----------------- CHẠY THỬ -----------------
# 1. Ví dụ TẠO HÌNH BÌNH HÀNH (Hình vuông)
# A=(0,0), B=(3,0), C=(3,3), D=(0,3)
A1 = (0, 0)
B1 = (3, 0)
C1 = (3, 3)
D1 = (0, 3)
print(f"Các điểm A1, B1, C1, D1 tạo thành HBH: {is_parallelogram(A1, B1, C1, D1)}")
# True
# 2. Ví dụ KHÔNG TẠO HÌNH BÌNH HÀNH (Hình thang)
# A=(0,0), B=(4,0), C=(3,2), D=(1,2)
A2 = (0, 0)
B2 = (4, 0)
C2 = (3, 2)
D2 = (1, 2)
print(f"Các điểm A2, B2, C2, D2 tạo thành HBH: {is_parallelogram(A2, B2, C2, D2)}")
# False
```

=> Kết quả sau khi chạy chương trình:

```
Các điểm A1, B1, C1, D1 tạo thành HBH: True
Các điểm A2, B2, C2, D2 tạo thành HBH: False
```

1.1.2. Code C++:

```
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
#include <cmath>             // Thư viện toán học
using namespace std;
// Ham tinh vector PQ = (xQ - xP, yQ - yP)
// P = (x1, y1), Q = (x2, y2)
pair<double, double> get_vector(pair<double,double> P, pair<double,double> Q)
{
    return make_pair(Q.first - P.first, Q.second - P.second);
}
// Kiem tra xem 4 diem A, B, C, D co tao thanh hinh binh hanh hay khong
// theo thu tu A -> B -> C -> D.
// Logic: Kiem tra xem cac vector doi co bang nhau khong:
// - Vector AB co bang Vector DC khong? (AB = DC)
// - Vector AD co bang Vector BC khong? (AD = BC)
// Luu y: Neu dung toa do so thuc (float), nen su dung sai so EPSILON
// de so sanh, nhung voi bai toan co ban nay, ta gia dinh so sanh truc tiep
// cac vector la du.
// Ham kiem tra hinh binh hanh
bool is_parallelogram(pair<double,double> A,
                      pair<double,double> B,
                      pair<double,double> C,
                      pair<double,double> D)
{
    // 1. Tinh cac vector can thiet
    pair<double,double> vector_AB = get_vector(A, B);  // (x_B - x_A, y_B - y_A)
    pair<double,double> vector_DC = get_vector(D, C);  // (x_C - x_D, y_C - y_D)
    pair<double,double> vector_AD = get_vector(A, D);  // (x_D - x_A, y_D - y_A)
    pair<double,double> vector_BC = get_vector(B, C);  // (x_C - x_B, y_C - y_B)
```

```
    // 2. Kiem tra dieu kien hinh binh hanh
    // Dieu kien 1: Vector AB phai bang Vector DC (AB || DC va |AB| = |DC|)
    bool condition1 = (vector_AB == vector_DC);
    // Dieu kien 2: Vector AD phai bang Vector BC (AD || BC va |AD| = |BC|)
    bool condition2 = (vector_AD == vector_BC);
    // Hinh binh hanh khi ca hai dieu kien thoa man
    return condition1 && condition2;
}
// ----------------- CHAY THU -----------------
int main()
{
    // 1. Vi du TAO HINH BINH HANH (Hinh vuong)
    // A=(0,0), B=(3,0), C=(3,3), D=(0,3)
    pair<double,double> A1 = {0, 0};
    pair<double,double> B1 = {3, 0};
    pair<double,double> C1 = {3, 3};
    pair<double,double> D1 = {0, 3};
    cout << "Cac diem A1, B1, C1, D1 tao thanh HBH: "
         << (is_parallelogram(A1, B1, C1, D1) ? "True" : "False") << endl;
    // 2. Vi du KHONG TAO HINH BINH HANH (Hinh thang)
    // A=(0,0), B=(4,0), C=(3,2), D=(1,2)
    pair<double,double> A2 = {0, 0};
    pair<double,double> B2 = {4, 0};
    pair<double,double> C2 = {3, 2};
    pair<double,double> D2 = {1, 2};
    cout << "Cac diem A2, B2, C2, D2 tao thanh HBH: "
         << (is_parallelogram(A2, B2, C2, D2) ? "True" : "False") << endl;
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
Các điểm A1, B1, C1, D1 tạo thành HBH: True
Các điểm A2, B2, C2, D2 tạo thành HBH: False
```

1.2. Bài tập 2 (Đa giác lồi): Cho N điểm. Kiểm tra xem 3 điểm bất kỳ có thẳng hàng hay không. Nếu không, hãy sắp xếp các điểm này theo thứ tự ngược chiều kim đồng hồ quanh điểm trung tâm của chúng.

1.2.1. Code Python:

```
import math
# Hằng số sai số nhỏ (Epsilon)
EPSILON = 1e-9
def cross_product(A, B, C):
    # Tính tích có hướng 2D của vector AB và vector AC.
    # (Bx - Ax) * (Cy - Ay) - (By - Ay) * (Cx - Ax)
    cross = (B[0] - A[0]) * (C[1] - A[1]) - \
            (B[1] - A[1]) * (C[0] - A[0])
    return cross
def is_collinear(A, B, C):
    # Kiểm tra ba điểm A, B, C có thẳng hàng hay không.
    cp = cross_product(A, B, C)
    return abs(cp) < EPSILON
def solve_convex_polygon_arrangement(points):
    # Giải quyết bài toán Đa giác lồi:
    # 1. Kiểm tra không có 3 điểm nào thẳng hàng.
    # 2. Sắp xếp các điểm ngược chiều kim đồng hồ quanh điểm neo (Anchor).
    N = len(points)
    if N < 3:
        return points # Không cần sắp xếp hoặc kiểm tra
```

```
    # --- Bước 1: Kiểm tra Thẳng hàng ---
    for i in range(N):
        for j in range(i + 1, N):
            for k in range(j + 1, N):
                if is_collinear(points[i], points[j], points[k]):
                    # Nếu tìm thấy 3 điểm thẳng hàng, trả về lỗi hoặc cảnh báo
                    print("Loi: Ton tai 3 diem thang hang. Khong phai da giac loi
đon gian.")
                    return None
    # --- Bước 2: Sắp xếp Ngược chiều Kim đồng hồ ---
    # 2a. Tìm điểm neo (Anchor point): Điểm có Y nhỏ nhất, nếu Y bằng nhau thì X
nhỏ nhất.
    # Điểm neo sẽ là điểm bắt đầu của chu trình ngược chiều kim đồng hồ.
    anchor = min(points, key=lambda p: (p[1], p[0]))
    # 2b. Sắp xếp các điểm còn lại dựa trên Góc so với điểm neo.
    # Ta sử dụng Tích có hướng để so sánh góc mà KHÔNG cần tính hàm math.atan2
(tránh lỗi số thực).
    def compare_points(P1, P2):
        # Hàm so sánh P1 và P2 dựa trên tích có hướng so với anchor.
        # Tạo điểm giả P_anchor để tính vector
        P_anchor = (anchor[0] + 1.0, anchor[1])
        # Tích có hướng của vector (anchor -> P1) và (anchor -> P2)
        # Kết quả > 0: P2 nằm bên trái (góc lớn hơn) so với P1 -> P1 trước P2
(Ngược chiều kim đồng hồ)
        cp = cross_product(anchor, P1, P2)
        # Xử lý trường hợp đặc biệt: P1 và P2 thẳng hàng với anchor
        if abs(cp) < EPSILON:
            # Nếu thẳng hàng, điểm gần anchor hơn sẽ được xếp trước.
            dist_sq = lambda P: (P[0] - anchor[0])**2 + (P[1] - anchor[1])**2
            dist1 = dist_sq(P1)
            dist2 = dist_sq(P2)
            # Nếu P1 gần hơn (dist1 < dist2), P1 đứng trước (-1)
            if dist1 < dist2: return -1
            else: return 1
        # So sánh dựa trên dấu của tích có hướng
        # cp > 0 => P2 nằm bên trái P1 (P1 trước P2) => trả về -1
        # cp < 0 => P2 nằm bên phải P1 (P2 trước P1) => trả về 1
        if cp > 0: return -1
        else: return 1
    # Tạo danh sách các điểm còn lại (loại bỏ anchor)
    sorted_points = [p for p in points if p != anchor]
    # Sắp xếp bằng hàm so sánh tùy chỉnh
    from functools import cmp_to_key
    sorted_points.sort(key=cmp_to_key(compare_points))
    # Đặt điểm neo về đầu
    sorted_points.insert(0, anchor)
    return sorted_points
# ----------------- CHẠY THỬ -----------------
# Bộ điểm ví dụ (Tạo thành hình vuông)
Points_Test = [
    (10.0, 10.0), # B
    (0.0, 0.0),   # A (Anchor)
    (10.0, 0.0),  # D
    (0.0, 10.0)   # C
]
print("--- Diem ban dau ---")
print(Points_Test)
Result = solve_convex_polygon_arrangement(Points_Test)
```

```
if Result:
    print("\n--- Ket qua Sap xep Nguoc chieu Kim dong ho ---")
    # Kết quả mong muốn: [(0.0, 0.0), (10.0, 0.0), (10.0, 10.0), (0.0, 10.0)]
    print(Result)
```

=> Kết quả sau khi chạy chương trình:

```
--- Diem ban dau ---
[(10.0, 10.0), (0.0, 0.0), (10.0, 0.0), (0.0, 10.0)]

--- Ket qua Sap xep Nguoc chieu Kim dong ho ---
[(0.0, 0.0), (10.0, 0.0), (10.0, 10.0), (0.0, 10.0)]
```

1.2.2. Code C++:

```
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
#include <cmath>             // Thư viện toán học
#include <algorithm>         // Thư viện để dùng sort
using namespace std;
// Hang so sai so nho (Epsilon)
const double EPSILON = 1e-9;
// Tinh tich co huong 2D cua vector AB va vector AC.
// (Bx - Ax) * (Cy - Ay) - (By - Ay) * (Cx - Ax)
double cross_product(const pair<double,double>& A,
                     const pair<double,double>& B,
                     const pair<double,double>& C)
{
    double cross = (B.first - A.first) * (C.second - A.second)
                 - (B.second - A.second) * (C.first - A.first);
    return cross;
}
// Kiem tra ba diem A, B, C co thang hang hay khong.
bool is_collinear(const pair<double,double>& A,
                  const pair<double,double>& B,
                  const pair<double,double>& C)
{
    double cp = cross_product(A, B, C);
    return fabs(cp) < EPSILON;
}
// Giai quyet bai toan Da giac loi:
// 1. Kiem tra khong co 3 diem nao thang hang.
// 2. Sap xep cac diem nguoc chieu kim dong ho quanh diem neo (Anchor).
vector<pair<double,double>>
solve_convex_polygon_arrangement(vector<pair<double,double>> points)
{
    int N = points.size();
    if (N < 3)
        return points; // Khong can sap xep hoac kiem tra
    // --- Buoc 1: Kiem tra Thang hang ---
    for (int i = 0; i < N; i++)
    {
        for (int j = i + 1; j < N; j++)
        {
            for (int k = j + 1; k < N; k++)
            {
                if (is_collinear(points[i], points[j], points[k]))
                {
                    // Neu tim thay 3 diem thang hang, tra ve loi hoac canh bao
                    cout << "Loi: Ton tai 3 diem thang hang. Khong phai da giac loi
don gian.\n";
```

```
                    return {};
                }
            }
        }
    }
    // --- Buoc 2: Sap xep Nguoc chieu Kim dong ho ---
    // 2a. Tim diem neo (Anchor point):
    // Diem co Y nho nhat, neu Y bang nhau thi X nho nhat.
    auto anchor = *min_element(points.begin(), points.end(),
                               [](auto& p1, auto& p2)
                               {
                                   if (fabs(p1.second - p2.second) < EPSILON)
                                       return p1.first < p2.first;
                                   return p1.second < p2.second;
                               });
    // 2b. Sap xep cac diem con lai dua tren Goc so voi diem neo.
    // Ta su dung Tich co huong de so sanh goc.
    auto compare_points = [&](const pair<double,double>& P1,
                              const pair<double,double>& P2)
    {
        // Tinh tich co huong cua vector (anchor -> P1) va (anchor -> P2)
        double cp = cross_product(anchor, P1, P2);
        // Xu ly truong hop dac biet: P1 va P2 thang hang voi anchor
        if (fabs(cp) < EPSILON)
        {
            auto dist_sq = [&](const pair<double,double>& P)
            {
                return (P.first - anchor.first)*(P.first - anchor.first)
                     + (P.second - anchor.second)*(P.second - anchor.second);
            };
            double dist1 = dist_sq(P1);
            double dist2 = dist_sq(P2);
            return dist1 < dist2; // diem gan hon dung truoc
        }
        // cp > 0 => P1 truoc P2 (nguoc chieu kim dong ho)
        return cp > 0;
    };
    // Tao danh sach cac diem con lai (loai bo anchor)
    vector<pair<double,double>> sorted_points;
    for (auto& p : points)
        if (!(fabs(p.first - anchor.first) < EPSILON &&
              fabs(p.second - anchor.second) < EPSILON))
            sorted_points.push_back(p);
    // Sap xep bang ham so sanh tuy chinh
    sort(sorted_points.begin(), sorted_points.end(), compare_points);
    // Dat anchor vao dau danh sach
    sorted_points.insert(sorted_points.begin(), anchor);
    return sorted_points;
}
// ----------------- CHAY THU -----------------
int main()
{
    // Bo diem vi du (Tao thanh hinh vuong)
    vector<pair<double,double>> Points_Test = {
        {10.0, 10.0}, // B
        {0.0, 0.0},   // A (Anchor)
        {10.0, 0.0},  // D
        {0.0, 10.0}   // C
```

```
    };
    cout << "--- Diem ban dau ---\n";
    for (auto& p : Points_Test)
        cout << "(" << p.first << ", " << p.second << ") ";
    cout << "\n";
    vector<pair<double,double>> Result =
solve_convex_polygon_arrangement(Points_Test);
    if (!Result.empty())
    {
        cout << "\n--- Ket qua Sap xep Nguoc chieu Kim dong ho ---\n";
        // Ket qua mong muon: (0,0), (10,0), (10,10), (0,10)
        for (auto& p : Result)
            cout << "(" << p.first << ", " << p.second << ") ";
        cout << "\n";
    }
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
--- Diem ban dau ---
[(10.0, 10.0), (0.0, 0.0), (10.0, 0.0), (0.0, 10.0)]

--- Ket qua Sap xep Nguoc chieu Kim dong ho ---
[(0.0, 0.0), (10.0, 0.0), (10.0, 10.0), (0.0, 10.0)]
```

2. Tổng kết, Dặn dò: Chuẩn bị các bài toán **xử lí xâu** cho buổi tiếp theo.

CHỦ ĐỀ 5. XỬ LÝ XÂU CƠ BẢN

BÀI 13. CÁC THAO TÁC CƠ BẢN VỚI XÂU

I. Mục Tiêu:

- **Kiến thức:** HS hiểu cách biểu diễn xâu trong Python (str) và C++ (std::string/char[]). Nắm vững các thao tác cơ bản: khai báo, truy cập, nối, tìm kiếm, và trích xuất chuỗi con.
- **Kỹ năng:** HS thành thạo việc cài đặt các thao tác xâu cơ bản và sử dụng các hàm/phương thức thư viện chuẩn.
- **Tư duy:** HS nhận biết sự khác biệt cơ bản trong cách quản lý bộ nhớ của xâu giữa hai ngôn ngữ (linh hoạt trong Python và thủ công hơn trong C++).

II. Phân Bổ Thời Gian (90 Phút):

Hoạt Động	Nội Dung	Thời gian (Phút)
Khởi động	Khái niệm Xâu, Biểu diễn trong Python/C++	10
Lý thuyết 1	Độ dài, Truy cập (Indexing) và Thay đổi Ký tự	20
Thực hành 1	Bài 1. Đảo ngược xâu, Đếm ký tự	20
Lý thuyết 2	Nối (Concatenation) và Trích xuất (Substring)	20
Thực hành 2	Bài 2. Tìm kiếm và Trích xuất Chuỗi con	15
Tổng kết	Bài tập về nhà, Củng cố	5
Tổng cộng		**90**

III. Lý Thuyết 1. Khai Báo, Độ dài và Truy cập (20 phút):

1. Khai báo và Độ dài:

Thao Tác	Python (str)	C++ (std::string)	Giải Thích
Khai báo	s = "Hello World"	string s = "Hello World";	Trong **C++**, luôn dùng **string** thay vì mảng **char[]** để dễ quản lý.
Độ dài	len(s)	s.length() hoặc s.size()	Cho biết số lượng ký tự. **C++** không tính ký tự kết thúc \0.

2. Truy cập và Thay đổi Ký tự (Indexing):

Xâu được coi là một mảng (hoặc danh sách) các ký tự, bắt đầu từ chỉ mục (index) **0**.

Thao Tác	Python	C++	Lưu ý quan trọng
Truy cập	s[i]	s[i] hoặc s.at(i)	**Python:** Hỗ trợ index âm (**s[-1]** là ký tự cuối cùng).
Thay đổi ký tự	**Không thể** (xâu bất biến/immutable)	s[i] = 'a';	**C++:** Xâu là biến đổi được (mutable).

IV. Thực Hành 1: Đảo ngược và Đếm Ký tự (20 phút):

1. Bài 1. Hàm Đếm ký tự và Đảo ngược Xâu:

- **Yêu cầu:** Viết hàm đếm số lần xuất hiện của một ký tự **c** trong xâu **s**. Viết hàm đảo ngược xâu **s**.

1.1. Code Python:

```python
# Đếm ký tự
def count_char(s, char):
    return s.count(char)
# Đảo ngược
def reverse_string(s):
    return s[::-1]
s = "programming"
print(f"Count 'g': {count_char(s, 'g')}")   # 2
print(f"Reverse: {reverse_string(s)}")      # gnimmargorp
```

=> Kết quả sau khi chạy chương trình:

```
Count 'g': 2
Reverse: gnimmargorp
```

- Giải thích Code Python:

+ **Đảo ngược:** Sử dụng kỹ thuật **slicing [::-1]** là cách Pythonic nhất.

+ **Đếm:** Dùng phương thức **count()** có sẵn.

1.2. Code C++:

```cpp
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
#include <string>            // Thư viện xử lí xâu
#include <algorithm>         // Thư viện dùng để đảo ngược xâu
```

```
using namespace std;
// 1. Hàm đếm số lần xuất hiện của một ký tự
int count_char(const string& s, char c)
{
    int count = 0;
    for (char ch : s)
    {
        if (ch == c)
        {
            count++;
        }
    }
    return count;
}
// 2. Hàm đảo ngược xâu
string reverse_string(string s)
{
    reverse(s.begin(), s.end());
    return s;
}
int main()
{
    string s = "programming";
    char char_to_count = 'g';
    int count = count_char(s, char_to_count);
    cout << "Count 'g': " << count << endl;     // 2
    string reversed_s = reverse_string(s);
    cout << "Reverse: " << reversed_s << endl;  // gnimmargorp
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
Count 'g': 2
Reverse: gnimmargorp
```

- Giải thích Code C++:

+ **Đảo ngược:** Cần thư viện **<algorithm>** và hàm **reverse(begin, end)**.

+ **Đếm:** Phải duyệt qua từng ký tự (hoặc dùng **count**).

V. Lý Thuyết 2: Nối và Trích xuất (20 phút)

1. Nối Xâu (Concatenation):

Thao Tác	Python	C++	Giải Thích
Nối	s1 = s2 + s3	string s1 = s2 + s3;	Toán tử + được sử dụng cho cả hai ngôn ngữ.
Nối tại chỗ	s1 += s2	s1 += s2;	Hiệu quả hơn so với tạo xâu mới.

2. Trích xuất Chuỗi con (Substring):

Thao Tác	Python	C++	Giải Thích
Trích xuất	s[start:end]	s.substr(pos, len)	**Python: [start:end]** không bao gồm ký tự tại **end**. **C++: len** là độ dài chuỗi con.

VI. Thực Hành 2: Tìm kiếm và Trích xuất (15 phút)

1. Bài 2. Tìm kiếm và Trích xuất Chuỗi con

- **Yêu cầu:** Cho xâu **s** và mẫu **p**. Tìm vị trí đầu tiên của **p** trong **s**. Nếu tìm thấy, trích xuất chuỗi con từ vị trí đó với độ dài của **p**.

1.1. Code Python:

```python
def find_and_extract(s, p):
    # Tìm vị trí đầu tiên của p trong s
    pos = s.find(p)
    if pos != -1:
        # Trích xuất chuỗi con s[pos : pos + len(p)]
        return s[pos : pos + len(p)]
    else:
        return "Not Found"
s = "banana_apple_banana"
p = "apple"
# 1. Trường hợp tìm thấy
print(f"Trich xuat 'apple': {find_and_extract(s, p)}")      # Kết quả: apple
# 2. Trường hợp không tìm thấy
print(f"Trich xuat 'key': {find_and_extract(s, 'key')}")    # Kết quả: Not Found
```

=> Kết quả sau khi chạy chương trình:

```
Trich xuat 'apple': apple
Trich xuat 'key': Not Found
```

- Giải thích Code Python:

+ **Tìm kiếm:** Dùng phương thức **s.find(p)**. Trả về **-1** nếu không tìm thấy.

+ **Trích xuất:** Dùng **slicing**, rất trực quan.

1.2. Code C++:

```cpp
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
#include <string>            // Thư viện xử lí xâu
using namespace std;         // Tiền tố std
string find_and_extract(const string& s, const string& p)
{
    // const string& s: Truyền xâu bằng tham chiếu hằng để tối ưu hiệu suất
    // Tìm vị trí đầu tiên của chuỗi con p trong s.
    // size_t là kiểu số nguyên không dấu, thường dùng cho kích thước/chỉ mục.
    size_t pos = s.find(p);
    // string::npos là một hằng số đặc biệt (thường là giá trị lớn nhất của size_t)
    // được trả về bởi find() khi không tìm thấy chuỗi con.
    if (pos != string::npos)
    {
        // Trích xuất chuỗi con:
        // s.substr(pos, p.length()): Bắt đầu tại pos, lấy độ dài bằng độ dài của
p.
        return s.substr(pos, p.length());
    }
    else
    {
        return "Not Found";
    }
}
int main()
{
    string s = "banana_apple_banana";
    string p1 = "apple";
```

```
    string p2 = "key";
    // 1. Trường hợp tìm thấy
    string result1 = find_and_extract(s, p1);
    cout << "Trich xuat '" << p1 << "': " << result1 << endl; // Kết quả: apple
    // 2. Trường hợp không tìm thấy
    string result2 = find_and_extract(s, p2);
    cout << "Trich xuat '" << p2 << "': " << result2 << endl; // Kết quả: Not Found
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
Trich xuat 'apple': apple
Trich xuat 'key': Not Found
```

- Giải thích Code C++:

+ **Tìm kiếm:** Dùng **s.find(p)**. So sánh kết quả với hằng số **string::npos**.

+ **Trích xuất:** Dùng phương thức **s.substr(pos, len)**.

VII. Bài Tập Về Nhà & Tổng Kết (5 phút):

1. Bài tập 1 (Xử lý chuỗi con): Cho một xâu **S** và hai chỉ số **i, j**. Đổi chỗ chuỗi con từ **S[i..j]** với chuỗi con từ **S[j+1..k]**. ***(Áp dụng Nối/Trích xuất)***.

1.1. Code Python:

```python
def swap_substrings(S, i, j, k):
    # Đổi chỗ chuỗi con S[i..j] với chuỗi con S[j+1..k].
    # Tham số:
    #   S (str): Xâu gốc.
    #   i (int): Chỉ số bắt đầu của chuỗi con 1 (S[i]).
    #   j (int): Chỉ số kết thúc của chuỗi con 1 (S[j]).
    #   k (int): Chỉ số kết thúc của chuỗi con 2 (S[k]).
    # Yêu cầu: 0 <= i <= j < j+1 <= k < len(S)
    # --- Bước 1: Trích xuất các phần của xâu ---
    # 1. Phần A (Prefix): S[0] đến S[i-1]
    # Cú pháp Python: S[:i]
    part_A = S[:i]
    # 2. Phần B (Substring 1): S[i] đến S[j]
    # Cú pháp Python: S[i] đến S[j] => S[i : j + 1]
    part_B = S[i : j + 1]
    # 3. Phần C (Substring 2): S[j+1] đến S[k]
    # Cú pháp Python: S[j+1] đến S[k] => S[j + 1 : k + 1]
    part_C = S[j + 1 : k + 1]
    # 4. Phần D (Suffix): S[k+1] đến hết
    # Cú pháp Python: S[k+1] đến hết => S[k + 1 :]
    part_D = S[k + 1 :]
    # --- Bước 2: Nối các phần theo thứ tự mới ---
    # Kết quả mong muốn: A + C + B + D
    new_S = part_A + part_C + part_B + part_D
    return new_S
# ----------------- CHẠY THỬ -----------------
S = "banana_apple_kiwi_orange"
# Ví dụ 1: Đổi chỗ "apple" và "kiwi"
# i=7 (a), j=11 (e) -> Chuỗi con 1: S[7..11] = "apple" (len 5)
# j+1=12 (_), k=15 (i) -> Chuỗi con 2: S[12..15] = "_kiw" (len 4)
# --> Ở đây ta giả định chỉ số k được chọn sao cho cả 2 chuỗi con cần đổi chỗ.
# Ta sẽ chọn k=16 để lấy được "_kiwi"
# Xâu S: "banana_apple_kiwi_orange" (len 24)
# Index: 012345678901234567890123
#           |    |     |     |
#           i    j    j+1    k
```

```
# Chọn i, j, k hợp lý:
i = 7    # Bắt đầu từ 'a'
j = 11   # Kết thúc ở 'e'
k = 16   # Kết thúc ở 'i'
print(f"Xau goc (S): {S}")
print(f"Chuoi con 1 (B): '{S[i:j+1]}'")     # apple
print(f"Chuoi con 2 (C): '{S[j+1:k+1]}'")   # _kiwi
S_new = swap_substrings(S, i, j, k)
# Xâu mới phải là: "banana" + "_kiwi" + "apple" + "_orange"
# Kết quả mong muốn: "banana_kiwi_apple_orange"
print(f"Xau sau khi doi: {S_new}")
```

=> Kết quả sau khi chạy chương trình:

```
Xau goc (S): banana_apple_kiwi_orange
Chuoi con 1 (B): 'apple'
Chuoi con 2 (C): '_kiwi'
Xau sau khi doi: banana__kiwiapple_orange
```

1.2. Code C++:

```
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
#include <string>            // Thư viện xử lí xâu
using namespace std;         // Tiền tố std
string swap_substrings(const string &S, int i, int j, int k)
{
    // Doi cho chuoi con S[i..j] voi chuoi con S[j+1..k].
    // Tham so:
    //   S (string): Xau goc.
    //   i (int): Chi so bat dau cua chuoi con 1 (S[i]).
    //   j (int): Chi so ket thuc cua chuoi con 1 (S[j]).
    //   k (int): Chi so ket thuc cua chuoi con 2 (S[k]).
    // Yeu cau: 0 <= i <= j < j+1 <= k < len(S)
    // --- Buoc 1: Trich xuat cac phan cua xau ---
    // 1. Phan A (Prefix): S[0] den S[i-1]
    // Cu phap tuong duong Python: S[:i]
    string part_A = S.substr(0, i);
    // 2. Phan B (Substring 1): S[i] den S[j]
    // Cu phap Python: S[i] den S[j] => S[i : j + 1]
    string part_B = S.substr(i, j - i + 1);
    // 3. Phan C (Substring 2): S[j+1] den S[k]
    // Cu phap Python: S[j+1] den S[k] => S[j + 1 : k + 1]
    string part_C = S.substr(j + 1, k - (j + 1) + 1);
    // 4. Phan D (Suffix): S[k+1] den het
    // Cu phap Python: S[k+1] den het => S[k + 1 :]
    string part_D = S.substr(k + 1);
    // --- Buoc 2: Noi cac phan theo thu tu moi ---
    // Ket qua mong muon: A + C + B + D
    string new_S = part_A + part_C + part_B + part_D;
    return new_S;
}
// ----------------- CHAY THU -----------------
int main()
{
    string S = "banana_apple_kiwi_orange";
    // Vi du 1: Doi cho "apple" va "kiwi"
    // i=7 (a), j=11 (e) -> Chuoi con 1: S[7..11] = "apple" (len 5)
    // j+1=12 (_), k=15 (i) -> Chuoi con 2: S[12..15] = "_kiw" (len 4)
    // --> O day ta gia dinh chi so k duoc chon sao cho ca 2 chuoi con can doi cho.
    // Ta se chon k=16 de lay duoc "_kiwi"
```

```
    // Xau S: "banana_apple_kiwi_orange" (len 24)
    // Index: 012345678901234567890123
    //           |    |     |     |
    //           i    j    j+1    k
    // Chon i, j, k hop ly:
    int i = 7;    // Bat dau tu 'a'
    int j = 11;   // Ket thuc o 'e'
    int k = 16;   // Ket thuc o 'i'
    cout << "Xau goc (S): " << S << "\n";
    cout << "Chuoi con 1 (B): '" << S.substr(i, j - i + 1) << "'\n";   // apple
    cout << "Chuoi con 2 (C): '" << S.substr(j + 1, k - (j + 1) + 1) << "'\n"; //
_kiwi
    string S_new = swap_substrings(S, i, j, k);
    // Xau moi phai la: "banana" + "_kiwi" + "apple" + "_orange"
    // Ket qua mong muon: "banana_kiwi_apple_orange"
    cout << "Xau sau khi doi: " << S_new << "\n";
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
Xau goc (S): banana_apple_kiwi_orange
Chuoi con 1 (B): 'apple'
Chuoi con 2 (C): '_kiwi'
Xau sau khi doi: banana__kiwiapple_orange
```

2. Bài tập 2 (Palindromic Check): Viết hàm kiểm tra xem một xâu có phải là xâu đối xứng (palindrome) hay không. ***(Áp dụng Đảo ngược và So sánh)***.

2.1. Code Python:

```python
def is_palindrome_by_reversal(S):
    # Kiểm tra xem xâu S có phải là xâu đối xứng (palindrome) hay không
    # bằng cách đảo ngược và so sánh.
    # Tham số:
    # S (str): Xâu cần kiểm tra.
    # --- Bước 1: Đảo ngược xâu ---
    # Sử dụng kỹ thuật slicing Pythonic: [::-1]
    S_reversed = S[::-1]
    # --- Bước 2: So sánh xâu gốc và xâu đảo ngược ---
    # Python cho phép so sánh trực tiếp hai xâu
    return S == S_reversed
# ----------------- CHẠY THỬ -----------------
S1 = "madam"
S2 = "racecar"
S3 = "hello"
S4 = "A man, a plan, a canal: Panama" # Có ký tự không phải chữ cái
print(f"'{S1}' co phai Palindrome khong? {is_palindrome_by_reversal(S1)}") # True
print(f"'{S2}' co phai Palindrome khong? {is_palindrome_by_reversal(S2)}") # True
print(f"'{S3}' co phai Palindrome khong? {is_palindrome_by_reversal(S3)}") # False
# Lưu ý: Khi làm bài toán Palindrome thực tế,
# ta thường bỏ qua khoảng trắng và ký tự đặc biệt, và chuyển về chữ thường.
# Hàm xử lý phức tạp hơn (bỏ qua khoảng trắng/đặc biệt và phân biệt chữ hoa/thường)
def is_palindrome_strict(S):
    import re
    # 1. Chuẩn hóa: Chuyển về chữ thường và loại bỏ các ký tự không phải chữ/số
    S_cleaned = re.sub(r'[^a-zA-Z0-9]', '', S).lower()
    # 2. So sánh với bản đảo ngược
    return S_cleaned == S_cleaned[::-1]
print(f"Kiem tra 'A man...' (Strict): {is_palindrome_strict(S4)}") # True
```

=> Kết quả sau khi chạy chương trình:

```
'madam' co phai Palindrome khong? True
'racecar' co phai Palindrome khong? True
'hello' co phai Palindrome khong? False
Kiem tra 'A man...' (Strict): True
```

2.2. Code C++:

```
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
#include <string>            // Thư viện xử lí xâu
using namespace std;         // Tiền tố std
bool is_palindrome_by_reversal(const string &S)
{
    // Kiem tra xem xau S co phai la xau doi xung (palindrome) hay khong
    // bang cach dao nguoc va so sanh.
    // Tham so:
    // S (string): Xau can kiem tra.
    // --- Buoc 1: Dao nguoc xau ---
    // Su dung ham reverse trong C++
    string S_reversed = S;
    reverse(S_reversed.begin(), S_reversed.end());
    // --- Buoc 2: So sanh xau goc va xau dao nguoc ---
    // C++ cho phep so sanh truc tiep hai xau
    return S == S_reversed;
}
bool is_palindrome_strict(const string &S)
{
    // Luu y: Khi lam bai toan Palindrome thuc te,
    // ta thuong bo qua khoang trang va ky tu dac biet, va chuyen ve chu thuong.
    // Ham xu ly phuc tap hon (bo qua khoang trang/dac biet va phan biet chu
hoa/thuong)
    // 1. Chuan hoa: Chuyen ve chu thuong va loai bo cac ky tu khong phai chu/so
    string S_cleaned;
    for (char c : S)
    {
        if (isalnum(c))
        {
            S_cleaned += tolower(c);
        }
    }
    // 2. So sanh voi ban dao nguoc
    string S_rev = S_cleaned;
    reverse(S_rev.begin(), S_rev.end());
    return S_cleaned == S_rev;
}
int main()
{
    // ----------------- CHAY THU -----------------
    string S1 = "madam";
    string S2 = "racecar";
    string S3 = "hello";
    string S4 = "A man, a plan, a canal: Panama"; // Co ky tu khong phai chu cai
    cout << "'" << S1 << "' co phai Palindrome khong? "
         << (is_palindrome_by_reversal(S1) ? "True" : "False") << "\n";
    cout << "'" << S2 << "' co phai Palindrome khong? "
         << (is_palindrome_by_reversal(S2) ? "True" : "False") << "\n";
    cout << "'" << S3 << "' co phai Palindrome khong? "
         << (is_palindrome_by_reversal(S3) ? "True" : "False") << "\n";
```

```
    cout << "Kiem tra 'A man...' (Strict): "
         << (is_palindrome_strict(S4) ? "True" : "False") << "\n";
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
'madam' co phai Palindrome khong? True
'racecar' co phai Palindrome khong? True
'hello' co phai Palindrome khong? False
Kiem tra 'A man...' (Strict): True
```

BÀI 14. ĐẾM TỪ & ĐẾM KÝ TỰ

I. Mục Tiêu:

- **Kiến thức:** HS nắm vững cách định nghĩa "từ" trong bối cảnh lập trình (chuỗi ký tự phân cách bởi khoảng trắng). Biết cách sử dụng các hàm thư viện chuẩn để phân tích xâu (tách xâu, kiểm tra loại ký tự).
- **Kỹ năng:** HS thành thạo việc cài đặt hàm đếm từ bằng phương pháp thủ công và bằng thư viện chuẩn (dùng **split()** trong **Python** và **stringstream** trong **C++**).
- **Tư duy:** HS rèn luyện kỹ năng xử lý các trường hợp đặc biệt (nhiều khoảng trắng liên tiếp, khoảng trắng ở đầu/cuối xâu).

II. Phân Bổ Thời Gian (90 Phút):

Hoạt Động	Nội Dung	Thời gian (Phút)
Ôn tập	Ôn tập Nối, Trích xuất, Đảo ngược xâu	10
Lý thuyết 1	Định nghĩa "Từ", Phương pháp đếm thủ công	15
Thực hành 1	Bài 1: Đếm Từ bằng phương pháp Sàng lọc/Thư viện	25
Lý thuyết 2	Các loại ký tự (isdigit, isalpha, isspace)	10
Thực hành 2	Bài 2: Thống kê số lượng Chữ cái, Số, Khoảng trắng	25
Tổng kết	Bài tập về nhà, Củng cố	5
Tổng cộng		**90**

III. Lý Thuyết 1: Đếm Từ (Word Counting) (15 phút)

1. Định nghĩa "Từ":

Trong lập trình cơ bản, một **từ** là một chuỗi ký tự liên tiếp được bao quanh bởi các ký tự **phân cách** (delimiters), chủ yếu là **khoảng trắng** (' '), tab (\t), hoặc xuống dòng (\n).

2. Hai Phương pháp Đếm Từ:

- **Phương pháp Thư viện (Tối ưu):** Sử dụng hàm/phương thức tách xâu (**split** trong **Python**, **stringstream** trong **C++**) để chia xâu thành các từ dựa trên khoảng trắng.
- **Phương pháp Thủ công (Duyệt trạng thái):** Duyệt từng ký tự, đếm sự chuyển đổi từ **ký tự phân cách** sang **ký tự chữ/số** (xác định bắt đầu một từ mới).

IV. Thực Hành 1: Đếm Từ (25 phút)

1. Bài 1. Hàm Đếm Từ trong Xâu

1.1. Code Python:

```python
def count_words_pythonic(S):
    # 1. Tách xâu S thành danh sách các từ.
    # Hàm split() tự động xử lý nhiều khoảng trắng liên tiếp
    # và khoảng trắng ở đầu/cuối xâu.
    words = S.split()
    # 2. Trả về số lượng từ
    return len(words)
S1 = " Hoc lap trinh Python "
S2 = "Mot.Hai.Ba" # Chỉ có khoảng trắng là delimiter
print(f"S1 words: {count_words_pythonic(S1)}") # 4
print(f"S2 words: {count_words_pythonic(S2)}") # 1
```

=> Kết quả sau khi chạy chương trình:

```
S1 words: 4
S2 words: 1
```

- Giải thích Code Python:

S.split(): Đây là cách Pythonic nhất. Nếu không có đối số, nó tự động dùng **bất kỳ chuỗi khoảng trắng** nào làm dấu phân cách, đồng thời **loại bỏ** các chuỗi rỗng.

1.2. Code C++:

```cpp
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
#include <string>            // Thư viện xử lí xâu
#include <sstream>           // Thư viện cho stringstream
using namespace std;         // Tiền tố std
int count_words_pythonic(const string &S)
{
    // 1. Tach xau S thanh danh sach cac tu.
    // Ham stringstream tu dong xu ly nhieu khoang trang lien tiep
    // va khoang trang o dau/cuoi xau.
    stringstream ss(S);
    string word;
    vector<string> words;
    while (ss >> word)
    {
        words.push_back(word);
    }
    // 2. Tra ve so luong tu
    return words.size();
}
int main()
{
    string S1 = " Hoc lap trinh Python ";
    string S2 = "Mot.Hai.Ba"; // Chi co khoang trang la delimiter
    cout << "S1 words: " << count_words_pythonic(S1) << "\n"; // 4
    cout << "S2 words: " << count_words_pythonic(S2) << "\n"; // 1
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
S1 words: 4
S2 words: 1
```

- Giải thích Code C++:

stringstream: Đây là phương pháp C++ chuẩn. Nó hoạt động tương tự như **split()** của Python, tự động bỏ qua khoảng trắng thừa và tách xâu hiệu quả.

V. Lý Thuyết 2: Phân loại và Đếm Ký tự (10 phút)

Để đếm các loại ký tự khác nhau (chữ cái, số, khoảng trắng...), ta cần sử dụng các hàm kiểm tra loại ký tự.

Thao Tác	Python	C++ (Thư viện <cctype>)	Ý Nghĩa
Kiểm tra Chữ cái	char.isalpha()	isalpha(char)	Là chữ cái (a-z, A-Z).
Kiểm tra Số	char.isdigit()	isdigit(char)	Là chữ số (0-9).
Kiểm tra Khoảng trắng	char.isspace()	isspace(char)	Là khoảng trắng (, \t, \n, v.v.).
Kiểm tra Chữ cái/Số	char.isalnum()	isalnum(char)	Là chữ cái hoặc chữ số.

VI. Thực Hành 2: Thống kê Loại Ký tự (25 phút)

1. Bài 2. Thống kê số lượng Chữ cái, Số, Khoảng trắng

- Yêu cầu: Viết hàm nhận vào xâu S và trả về số lượng chữ cái, chữ số, và ký tự khoảng trắng.

1.1. Code Python:

```python
def analyze_string(S):
    counts = {'letters': 0, 'digits': 0, 'spaces': 0}
    for char in S:
        if char.isalpha():
            counts['letters'] += 1
        elif char.isdigit():
            counts['digits'] += 1
        elif char.isspace():
            counts['spaces'] += 1
    return counts
S = "Phuong phap hoc nam 2025."
stats = analyze_string(S)
print(f"Letters: {stats['letters']}")   # 16
print(f"Digits: {stats['digits']}")     # 4
print(f"Spaces: {stats['spaces']}")     # 4
```

=> Kết quả sau khi chạy chương trình:

```
Letters: 16
Digits: 4
Spaces: 4
```

- Giải thích Code Python:

Phương thức Xâu: Python cho phép gọi trực tiếp các phương thức kiểm tra trên đối tượng ký tự (**char.isalpha()**). Sử dụng **dict** để trả về thống kê dễ dàng.

1.2. Code C++:

```cpp
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
#include <string>            // Thư viện xử lí xâu
#include <sstream>           // Thư viện cho stringstream
```

```
#include <cctype>           // Thư viện cho các hàm is...
using namespace std;        // Tiền tố std
struct Counts
{
    int letters;
    int digits;
    int spaces;
};
Counts analyze_string(const string &S)
{
    Counts counts = {0, 0, 0};
    for (char char_ : S)
    {
        if (isalpha(char_))
        {
            counts.letters += 1;
        } else if (isdigit(char_))
        {
            counts.digits += 1;
        } else if (isspace(char_))
        {
            counts.spaces += 1;
        }
    }
    return counts;
}
int main()
{
    string S = "Phuong phap hoc nam 2025.";
    Counts stats = analyze_string(S);
    cout << "Letters: " << stats.letters << "\n";   // 16
    cout << "Digits: " << stats.digits << "\n";     // 4
    cout << "Spaces: " << stats.spaces << "\n";     // 4
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
Letters: 16
Digits: 4
Spaces: 4
```

- Giải thích Code C++:

Thư viện <cctype>: Cần **include** thư viện này. Các hàm như **isalpha()**, **isdigit()**, **isspace()** nhận ký tự **char** làm đối số. Sử dụng **struct** để trả về nhiều giá trị thống kê.

VII. Bài Tập Về Nhà & Tổng Kết (5 phút)

1. Bài tập 1 (Đếm Từ Thủ công): Viết lại hàm **count_words** bằng phương pháp duyệt thủ công (không dùng **split** hoặc **stringstream**), chỉ dựa vào việc kiểm tra sự chuyển đổi từ khoảng trắng sang ký tự thường.

1.1. Code Python:

```
import string   # Thư viện xử lí xâu
# Định nghĩa các trạng thái
OUT_OF_WORD = 0     # Trạng thái ban đầu: Đang nằm trong vùng khoảng trắng
IN_WORD = 1         # Trạng thái: Đang nằm trong một từ
def count_words_manual(S):
    # Đếm số lượng từ trong xâu S bằng phương pháp duyệt thủ công (Duyệt trạng
thái).
    # Một từ được định nghĩa là chuỗi ký tự không phải khoảng trắng.
```

```
    # Hàm này xử lý khoảng trắng ở đầu, cuối và giữa xâu.
    state = OUT_OF_WORD
    word_count = 0
    # Ký tự phân cách: khoảng trắng, tab, newline
    # Trong Python, char.isspace() kiểm tra các loại này
    for char in S:
        # Kiểm tra: Ký tự hiện tại có phải khoảng trắng không?
        if char.isspace():
            # Nếu là khoảng trắng, chuyển về trạng thái OUT_OF_WORD (hoặc duy trì)
            state = OUT_OF_WORD
        else:
            # Nếu là ký tự chữ/số (không phải khoảng trắng)
            # 1. Kiểm tra sự chuyển trạng thái: Từ OUT_OF_WORD sang IN_WORD
            if state == OUT_OF_WORD:
                # Đây là ký tự đầu tiên của một từ mới
                word_count += 1
                # Chuyển trạng thái sang IN_WORD
                state = IN_WORD
            # Nếu state đã là IN_WORD, ta tiếp tục duyệt từ đó (không làm
gì)
    return word_count
# ----------------- CHẠY THỬ -----------------
S1 = "  Hoc lap trinh thu cong "
S2 = "Mot.Hai.Ba"
S3 = "\t\n  Test \t\n"
print(f"'{S1}' |So tu thu cong: {count_words_manual(S1)}") # 4
print(f"'{S2}' |So tu thu cong: {count_words_manual(S2)}") # 1 (vì '.' không phải
space)
print(f"'{S3}' |So tu thu cong: {count_words_manual(S3)}") # 1
```

=> Kết quả sau khi chạy chương trình:

```
'  Hoc lap trinh thu cong ' |So tu thu cong: 5
'Mot.Hai.Ba' |So tu thu cong: 1
'

  Test
' |So tu thu cong: 1
```

1.2. Code C++:

```
#include <iostream>         // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>    // Thư viện tổng hợp
#include <string>           // Thư viện xử lí xâu
#include <sstream>          // Thư viện cho stringstream
#include <cctype>           // Thư viện cho các hàm is...
using namespace std;        // Tiền tố std
// Dinh nghia cac trang thai
const int OUT_OF_WORD = 0;   // Trang thai ban dau: Dang nam trong vung khoang
trang
const int IN_WORD = 1;       // Trang thai: Dang nam trong mot tu
int count_words_manual(const string &S)
{
    // Dem so luong tu trong xau S bang phuong phap duyet thu cong (Duyet trang
thai).
    // Mot tu duoc dinh nghia la chuoi ky tu khong phai khoang trang.
    // Ham nay xu ly khoang trang o dau, cuoi va giua xau.
    int state = OUT_OF_WORD;
    int word_count = 0;
    // Ky tu phan cach: khoang trang, tab, newline
    // Trong C++, isspace() kiem tra cac loai ky tu trang tuong tu Python
    for (char ch : S)
```

```
    {
        // Kiem tra: Ky tu hien tai co phai khoang trang khong?
        if (isspace(ch))
        {
            // Neu la khoang trang, chuyen ve trang thai OUT_OF_WORD (hoac duy tri)
            state = OUT_OF_WORD;
        }
        else
        {
            // Neu la ky tu chu/so (khong phai khoang trang)
            // 1. Kiem tra su chuyen trang thai: Tu OUT_OF_WORD sang IN_WORD
            if (state == OUT_OF_WORD)
            {
                // Day la ky tu dau tien cua mot tu moi
                word_count += 1;
                // Chuyen trang thai sang IN_WORD
                state = IN_WORD;
            }
            // Neu state da la IN_WORD, ta tiep tuc duyet tu do (khong lam gi)
        }
    }
    return word_count;
}
// ----------------- CHAY THU -----------------
int main()
{
    string S1 = "  Hoc lap trinh thu cong ";
    string S2 = "Mot.Hai.Ba";
    string S3 = "\t\n  Test \t\n";
    cout << "'" << S1 << "' |So tu thu cong: " << count_words_manual(S1) << "\n";
// 4
    cout << "'" << S2 << "' |So tu thu cong: " << count_words_manual(S2) << "\n";
// 1 (vi '.' khong phai space)
    cout << "'" << S3 << "' |So tu thu cong: " << count_words_manual(S3) << "\n";
// 1
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
'  Hoc lap trinh thu cong ' |So tu thu cong: 5
'Mot.Hai.Ba' |So tu thu cong: 1
'
  Test
' |So tu thu cong: 1
```

2. Bài tập 2 (Tần suất Ký tự): Viết chương trình tính tần suất (số lần xuất hiện) của **mọi** ký tự trong xâu, sau đó in ra ký tự xuất hiện nhiều nhất. (Sử dụng **Dictionary** trong **Python** hoặc **std::map/mảng** trong C++).

2.1. Code Python:

```
def analyze_character_frequency_loop_manual(S):
    # Tính tần suất của mọi ký tự và tìm ký tự xuất hiện nhiều nhất
    # bằng cách sử dụng vòng lặp thủ công.
    # Tham số:
    # S (str): Xâu đầu vào.
    # Trả về:
    # tuple: (Dictionary tần suất, Ký tự xuất hiện nhiều nhất, Tần suất lớn
nhất)
```

```python
    # --- Bước 1: Tính Tần suất bằng Dictionary (Thủ công) ---
    frequency = {}
    for char in S:
        # Nếu ký tự đã có, tăng lên 1. Nếu chưa có, khởi tạo là 1.
        frequency[char] = frequency.get(char, 0) + 1
    # --- Bước 2: Tìm Ký tự Xuất hiện Nhiều nhất bằng Vòng lặp ---
    if not frequency:
        # Xử lý xâu rỗng
        return {}, None, 0
    most_common_char = None
    max_frequency = -1 # Khởi tạo tần suất lớn nhất là -1
    # Duyệt qua các cặp (ký tự, tần suất) trong dictionary
    for char, freq in frequency.items():
        # Kiểm tra: Nếu tần suất hiện tại lớn hơn tần suất lớn nhất đã tìm thấy
        if freq > max_frequency:
            max_frequency = freq         # Cập nhật tần suất lớn nhất
            most_common_char = char      # Lưu lại ký tự tương ứng
    return frequency, most_common_char, max_frequency
# ----------------- CHẠY THỬ -----------------
S1 = "programming language python"
S2 = "aabbcccccccdddd"
# 1. Kiểm tra S1
freq1, max_char1, max_freq1 = analyze_character_frequency_loop_manual(S1)
print(f"--- Phan tich Xau 1: '{S1}' ---")
print("1. Tan suat (Dictionary):", freq1)
print(f"2. Ky tu xuat hien nhieu nhat: '{max_char1}' ({max_freq1} lan)")
# 2. Kiểm tra S2
freq2, max_char2, max_freq2 = analyze_character_frequency_loop_manual(S2)
print(f"\n--- Phan tich Xau 2: '{S2}' ---")
print("1. Tan suat (Dictionary):", freq2)
print(f"2. Ky tu xuat hian nhieu nhat: '{max_char2}' ({max_freq2} lan)")
```

=> Kết quả sau khi chạy chương trình:

```
--- Phan tich Xau 1: 'programming language python' ---
1. Tan suat (Dictionary): {'p': 2, 'r': 2, 'o': 2, 'g': 4, 'a': 3, 'm': 2, 'i': 1, 'n': 3, ' ': 2, 'l': 1, 'u': 1, 'e': 1,
'y': 1, 't': 1, 'h': 1}
2. Ky tu xuat hien nhieu nhat: 'g' (4 lan)

--- Phan tich Xau 2: 'aabbcccccccdddd' ---
1. Tan suat (Dictionary): {'a': 2, 'b': 2, 'c': 7, 'd': 4}
2. Ky tu xuat hian nhieu nhat: 'c' (7 lan)
```

2.2. Code C++:

```cpp
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
#include <string>            // Thư viện xử lí xâu
using namespace std;         // Tiền tố std
// Ham phan tich tan suat ky tu bang vong lap thu cong
tuple<map<char, int>, char, int> analyze_character_frequency_loop_manual(const
string &S)
{
    // Tinh tan suat cua moi ky tu va tim ky tu xuat hien nhieu nhat
    // bang cach su dung vong lap thu cong.
    // Tham so:
    // S (str): Xau dau vao.
    // Tra ve:
    // tuple: (Dictionary tan suat, Ky tu xuat hien nhieu nhat, Tan suat lon nhat)
    // --- Buoc 1: Tinh Tan suat bang Dictionary (Thu cong) ---
    map<char, int> frequency;
    for (char charA : S)
    {
```

```
        // Neu ky tu da co, tang len 1. Neu chua co, khoi tao la 1.
        frequency[charA] = frequency[charA] + 1;
    }
    // --- Buoc 2: Tim Ky tu Xuat hien Nhieu nhat bang Vong lap ---
    if (frequency.empty())
    {
        // Xu ly xau rong
        return make_tuple(map<char, int>(), (char)0, 0);
    }
    char most_common_char = 0;
    int max_frequency = -1; // Khoi tao tan suat lon nhat la -1
    // Duyet qua cac cap (ky tu, tan suat) trong dictionary
    for (auto &p : frequency)
    {
        char charA = p.first;
        int freq = p.second;
        // Kiem tra: Neu tan suat hien tai lon hon tan suat lon nhat da tim thay
        if (freq > max_frequency)
        {
            max_frequency = freq;          // Cap nhat tan suat lon nhat
            most_common_char = charA;      // Luu lai ky tu tuong ung
        }
    }
    return make_tuple(frequency, most_common_char, max_frequency);
}
// ----------------- CHAY THU -----------------
int main()
{
    string S1 = "programming language python";
    string S2 = "aabbcccccccdddd";
    // 1. Kiem tra S1
    auto [freq1, max_char1, max_freq1] =
analyze_character_frequency_loop_manual(S1);
    cout << "--- Phan tich Xau 1: '" << S1 << "' ---\n";
    cout << "1. Tan suat (Dictionary): ";
    for (auto &p : freq1) cout << "{" << p.first << ": " << p.second << "} ";
    cout << "\n2. Ky tu xuat hien nhieu nhat: '" << max_char1 << "' (" << max_freq1
<< " lan)\n";
    // 2. Kiem tra S2
    auto [freq2, max_char2, max_freq2] =
analyze_character_frequency_loop_manual(S2);
    cout << "\n--- Phan tich Xau 2: '" << S2 << "' ---\n";
    cout << "1. Tan suat (Dictionary): ";
    for (auto &p : freq2) cout << "{" << p.first << ": " << p.second << "} ";
    cout << "\n2. Ky tu xuat hian nhieu nhat: '" << max_char2 << "' (" << max_freq2
<< " lan)\n";
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
--- Phan tich Xau 1: 'programming language python' ---
1. Tan suat (Dictionary): { : 2} {a: 3} {e: 1} {g: 4} {h: 1} {i: 1} {l: 1} {m: 2} {n: 3} {o: 2} {p: 2} {r: 2} {t: 1} {u: 1}
 {y: 1}
2. Ky tu xuat hien nhieu nhat: 'g' (4 lan)

--- Phan tich Xau 2: 'aabbcccccccdddd' ---
1. Tan suat (Dictionary): {a: 2} {b: 2} {c: 7} {d: 4}
2. Ky tu xuat hian nhieu nhat: 'c' (7 lan)
```

BÀI 15. BÀI TOÁN XÂU CON

I. Mục Tiêu:

- **Kiến thức:** HS hiểu rõ sự khác biệt giữa chuỗi con (**substring**) và chuỗi con không liên tục (**subsequence**). Nắm vững kỹ thuật duyệt qua tất cả các chuỗi con của một xâu.
- **Kỹ năng:** HS thành thạo việc cài đặt các vòng lặp lồng nhau để trích xuất/kiểm tra tất cả các chuỗi con. Áp dụng kỹ thuật **"cửa sổ trượt" (Sliding Window)** cho các bài toán tối ưu.
- **Tư duy:** Rèn luyện kỹ năng phân tích độ phức tạp thời gian ($O(N^2)$, $O(N^3)$) khi duyệt qua chuỗi con.

II. Phân Bổ Thời Gian (90 Phút):

Hoạt Động	Nội Dung	Thời gian (Phút)
Ôn tập	Ôn tập Tần suất ký tự, Đếm từ	10
Lý thuyết 1	Chuỗi con (Substring) và Chuỗi con không liên tục (Subsequence)	10
Thực hành 1	Bài 1. Liệt kê và Duyệt tất cả Chuỗi con	20
Lý thuyết 2	Bài toán Chuỗi con đối xứng (Palindrome Substring)	15
Thực hành 2	Bài 2. Tìm Chuỗi con đối xứng dài nhất	30
Tổng kết	Bài tập về nhà, Củng cố	5
Tổng cộng		**90**

III. Lý Thuyết 1: Khái niệm về Chuỗi con (10 phút)

1. Chuỗi con (Substring):

- **Định nghĩa:** Là một phần **liên tục** của xâu gốc S.
- **Ví dụ:** Xâu S = "banana". Các chuỗi con là: "b", "a", "ban", "anan", "banana".

2. Chuỗi con không liên tục (Subsequence):

- **Định nghĩa:** Là một chuỗi ký tự được tạo thành bằng cách xóa **không liên tục** (hoặc không xóa) các ký tự khỏi xâu gốc.
- **Ví dụ:** Xâu S = "banana". Các chuỗi con không liên tục là: "bna", "bnn".
- **Trong bài này, ta chỉ tập trung vào Chuỗi con (Substring).**

3. Kỹ thuật Duyệt tất cả Chuỗi con:

- Một xâu S có độ dài N có N(N+1)/2 chuỗi con. Ta cần hai vòng lặp lồng nhau $O(N^2)$ để duyệt qua tất cả:
  + Vòng lặp ngoài (i): Chỉ mục bắt đầu.
  + Vòng lặp trong (j): Chỉ mục kết thúc.

IV. Thực Hành 1: Liệt kê Chuỗi con (20 phút)

1. Bài 1. Liệt kê tất cả Chuỗi con của một Xâu

1.1. Code Python:

```
def list_all_substrings(S):
    N = len(S)
    substrings = []
```

```
    for i in range(N):
        # i: Bắt đầu (0 to N-1)
        for j in range(i, N):
            # j: Kết thúc (i to N-1)
            # Slicing: S[i : j+1] lấy từ i đến j
            sub = S[i : j + 1]
            substrings.append(sub)
    return substrings
S = "abc"
# Kết quả: ['a', 'ab', 'abc', 'b', 'bc', 'c']
print(list_all_substrings(S))
```

=> Kết quả sau khi chạy chương trình:

```
['a', 'ab', 'abc', 'b', 'bc', 'c']
```

- Giải thích Code Python:

Slicing Python: Rất tiện lợi, S[i : j + 1] trích xuất trực tiếp chuỗi con. Độ phức tạp tổng thể là $O(N^2)$ lần trích xuất, mỗi lần trích xuất mất O(N), tổng là $O(N^3)$ nếu tính cả chi phí tạo chuỗi con mới. Nếu chỉ duyệt, là $O(N^2)$.

1.2. Code C++:

```
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
#include <string>            // Thư viện xử lí xâu
#include <vector>            // Thư viện xử lí vector
using namespace std;         // Tiền tố std
vector<string> list_all_substrings(string S)
{
    int N = S.length();
    vector<string> substrings;
    for (int i = 0; i < N; i++)
    {
        // i: Bat dau (0 to N-1)
        for (int j = i; j < N; j++)
        {
            // j: Ket thuc (i to N-1)
            // Slicing: S[i : j+1] lay tu i den j
            string sub = S.substr(i, j - i + 1);
            substrings.push_back(sub);
        }
    }
    return substrings;
}
int main()
{
    string S = "abc";
    // Ket qua: ['a', 'ab', 'abc', 'b', 'bc', 'c']
    vector<string> result = list_all_substrings(S);
    for (auto &x : result)
    {
        cout << x << " ";
    }
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
a ab abc b bc c
```

- Giải thích Code C++:

substr(pos, len): C++ yêu cầu độ dài (len). Tương tự Python, độ phức tạp $O(N^3)$ nếu tính cả chi phí tạo chuỗi con mới, hoặc $O(N^2)$ nếu chỉ duyệt.

V. Lý thuyết 2: Bài toán Chuỗi con Đối xứng (15 phút)

1. Định nghĩa:

Chuỗi con Đối xứng (Palindromic Substring) là một chuỗi con của xâu S mà bản thân nó là một xâu đối xứng.

2. Kỹ thuật "Mở rộng từ tâm" (Expand Around Center):

Đây là phương pháp tối ưu hơn so với việc duyệt $O(N^2)$ chuỗi con và kiểm tra đối xứng O(N) (tổng $O(N^3)$).

- **Bước 1.** Duyệt qua tất cả 2N – 1 **tâm** có thể có trong xâu S.
  + N tâm là các ký tự đơn (ví dụ: a**b**a).
  + N – 1 tâm là các khoảng trống giữa hai ký tự (ví dụ: ab|ba).
- **Bước 2.** Từ mỗi tâm, mở rộng sang hai bên, kiểm tra xem hai ký tự đối xứng có bằng nhau không.

=> **Độ phức tạp:** $O(N^2)$ (vì có O(N) tâm, mỗi lần mở rộng tối đa O(N) lần).

VI. Thực Hành 2: Chuỗi con Đối xứng dài nhất (30 phút):

1. Bài 2. Tìm Chuỗi con Đối xứng dài nhất

1.1. Code Python:

```python
def expand_center(S, left, right):
    # Mở rộng từ tâm (left, right) và trả về độ dài đối xứng
    while left >= 0 and right < len(S) and S[left] == S[right]:
        left -= 1
        right += 1
        # Trả về độ dài đối xứng được tìm thấy
    return right - left - 1
def longest_palindromic_substring(S):
    if len(S) < 2: return S
    start = 0   # Chỉ mục bắt đầu của chuỗi con dài nhất
    max_len = 0 # Độ dài dài nhất
    for i in range(len(S)):
        # 1. Tâm lẻ: Ký tự đơn (ví dụ: 'a'ba). left=right=i
        len1 = expand_center(S, i, i)
        # 2. Tâm chẵn: Khoảng trống (ví dụ: 'aa'). left=i, right=i+1
        len2 = expand_center(S, i, i + 1)
        current_max_len = max(len1, len2)
        if current_max_len > max_len:
            max_len = current_max_len
            # Cập nhật vị trí bắt đầu mới:
            # start = i - (max_len - 1) // 2
            start = i - (current_max_len - 1) // 2
    return S[start : start + max_len]
S = "babad"
print(f"LPS of 'babad': {longest_palindromic_substring(S)}") # bab/aba
```

=> Kết quả sau khi chạy chương trình:

```
LPS of 'babad': bab
```

- Giải thích Code Python:

Tính start: Chỉ số bắt đầu được tính toán từ tâm **i** và độ dài đối xứng (**max_len**). Công thức này đảm bảo lấy đúng vị trí bắt đầu, dù tâm là lẻ hay chẵn.

1.2. Code C++:

```
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
#include <string>            // Thư viện xử lí xâu
using namespace std;         // Tiền tố std
int expand_center(const string &S, int left, int right)
{
    // Mo rong tu tam (left, right) va tra ve do dai doi xung
    while (left >= 0 && right < (int)S.length() && S[left] == S[right])
    {
        left -= 1;
        right += 1;
    }
    // Tra ve do dai doi xung duoc tim thay
    return right - left - 1;
}
string longest_palindromic_substring(const string &S)
{
    if (S.length() < 2) return S;
    int start = 0;   // Chi muc bat dau cua chuoi con dai nhat
    int max_len = 0; // Do dai dai nhat
    for (int i = 0; i < (int)S.length(); i++)
    {
        // 1. Tam le: Ky tu don (vi du: 'a'ba). left=right=i
        int len1 = expand_center(S, i, i);
        // 2. Tam chan: Khoang trong (vi du: 'aa'). left=i, right=i+1
        int len2 = expand_center(S, i, i + 1);
        int current_max_len = max(len1, len2);
        if (current_max_len > max_len)
        {
            max_len = current_max_len;
            // Cap nhat vi tri bat dau moi:
            // start = i - (max_len - 1) // 2
            start = i - (current_max_len - 1) / 2;
        }
    }
    return S.substr(start, max_len);
}
int main()
{
    string S = "babad";
    cout << "LPS of 'babad': " << longest_palindromic_substring(S); // bab/aba
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
LPS of 'babad': bab
```

- Giải thích Code C++:

Tính start: Tương tự Python, sử dụng phép chia số nguyên (/ 2) để tìm điểm bắt đầu chính xác. Cần sử dụng **substr(start, max_len)** để trích xuất kết quả.

VII. Bài Tập Về Nhà & Tổng Kết (5 phút):

1. Bài tập 1 (Chuỗi con Đơn nhất): Tìm số lượng chuỗi con **chỉ chứa** các ký tự khác nhau (**distinct characters**). ***(Áp dụng kỹ thuật cửa sổ trượt).***

1.1. Code Python:

```
def count_distinct_substrings(S):
    # Tìm số lượng chuỗi con chỉ chứa các ký tự khác nhau (distinct characters)
    # bằng kỹ thuật Cửa sổ trượt (Sliding Window).
    # Độ phức tạp: O(N)
    N = len(S)
    if N == 0:
        return 0
    left = 0
    total_count = 0
    # Set để lưu trữ các ký tự duy nhất trong cửa sổ S[left...right]
    char_set = set()
    # Con trỏ 'right' duyệt qua từng ký tự của xâu (mở rộng cửa sổ)
    for right in range(N):
        # --- BƯỚC THU HẸP (Shrink) ---
        # Nếu ký tự S[right] đã tồn tại trong Set, nghĩa là có trùng lặp.
        # Ta cần thu hẹp cửa sổ từ bên trái (tăng 'left') cho đến khi ký tự trùng
lặp
        # bị loại bỏ khỏi cửa sổ hiện tại.
        while S[right] in char_set:
            char_set.remove(S[left])
            left += 1
        # --- BƯỚC MỞ RỘNG (Expand) ---
        # Ký tự S[right] hiện tại là duy nhất trong cửa sổ S[left...right]
        char_set.add(S[right])
        # --- BƯỚC TÍNH TOÁN KẾT QUẢ ---
        # Độ dài cửa sổ hợp lệ hiện tại: L = right - left + 1
        # Tại vị trí 'right', có L chuỗi con đơn nhất kết thúc tại 'right'.
        # Ví dụ: S[left..right] = "abc". Kết thúc tại 'c', ta có: "c", "bc", "abc".
        window_length = right - left + 1
        total_count += window_length
    return total_count
# ----------------- CHẠY THỬ -----------------
S1 = "aab"
# Chuỗi con: a, a, b, aa (X), ab (X), aab (X)
# Chuỗi con đơn nhất: a, b. Tổng 3 (a, b, a)
# Phân tích:
# r=0 (a): set={a}. len=1. count+=1. total=1 (chuỗi "a")
# r=1 (a): S[1] trùng. Thu hẹp: loại S[0]='a'. left=1. set={}. Thêm S[1]='a'.
set={a}. len=1. count+=1. total=2 (chuỗi "a")
# r=2 (b): set={a, b}. len=2. count+=2. total=4 (chuỗi "b", "ab")
# Tổng cộng: 4 chuỗi con đơn nhất: "a", "b", "a", "ab"
S2 = "abc"
# Chuỗi con đơn nhất: a, b, c, ab, bc, abc. Tổng 6.
# Phân tích: r=0 (1), r=1 (2), r=2 (3). Total: 1+2+3 = 6
S3 = "bbbbb"
# Chuỗi con đơn nhất: b, b, b, b, b. Tổng 5.
# Phân tích: r=0 (1). r=1 (trùng, loại S[0]). len=1. total=2. r=2 (trùng, loại
S[1]). len=1. total=3. ...
print(f"S='{S1}' | So luong chuoi con don nhat: {count_distinct_substrings(S1)}") #
4
print(f"S='{S2}' | So luong chuoi con don nhat: {count_distinct_substrings(S2)}") #
6
print(f"S='{S3}' | So luong chuoi con don nhat: {count_distinct_substrings(S3)}") #
5
```

=> Kết quả sau khi chạy chương trình:

```
S='aab' | So luong chuoi con don nhat: 4
S='abc' | So luong chuoi con don nhat: 6
S='bbbbb' | So luong chuoi con don nhat: 5
```

1.2. Code C++:

```
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
#include <string>            // Thư viện xử lí xâu
using namespace std;         // Tiền tố std
int count_distinct_substrings(const string &S)
{
    // Tim so luong chuoi con chi chua cac ky tu khac nhau (distinct characters)
    // bang ky thuat Cua so truot (Sliding Window).
    // Do phuc tap: O(N)
    int N = S.length();
    if (N == 0)
    {
        return 0;
    }
    int left = 0;
    int total_count = 0;
    // Set de luu tru cac ky tu duy nhat trong cua so S[left...right]
    unordered_set<char> char_set;
    // Con tro 'right' duyet qua tung ky tu cua xau (mo rong cua so)
    for (int right = 0; right < N; right++)
    {
        // --- BUOC THU HEP (Shrink) ---
        // Neu ky tu S[right] da ton tai trong Set, nghia la co trung lap.
        // Ta can thu hep cua so tu ben trai (tang 'left') cho den khi ky tu trung
lap
        // bi loai bo khoi cua so hien tai.
        while (char_set.find(S[right]) != char_set.end())
        {
            char_set.erase(S[left]);
            left += 1;
        }
        // --- BUOC MO RONG (Expand) ---
        // Ky tu S[right] hien tai la duy nhat trong cua so S[left...right]
        char_set.insert(S[right]);
        // --- BUOC TINH TOAN KET QUA ---
        // Do dai cua so hop le hien tai: L = right - left + 1
        // Tai vi tri 'right', co L chuoi con don nhat ket thuc tai 'right'.
        // Vi du: S[left..right] = "abc". Ket thuc tai 'c', ta co: "c", "bc",
"abc".
        int window_length = right - left + 1;
        total_count += window_length;
    }
    return total_count;
}
int main()
{
    // ----------------- CHAY THU -----------------
    string S1 = "aab";
    // Chuoi con: a, a, b, aa (X), ab (X), aab (X)
    // Chuoi con don nhat: a, b. Tong 3 (a, b, a)
    // Phan tich:
    // r=0 (a): set={a}. len=1. count+=1. total=1 (chuoi "a")
```

```
    // r=1 (a): S[1] trung. Thu hep: loai S[0]='a'. left=1. set={}. Them S[1]='a'.
set={a}. len=1. count+=1. total=2 (chuoi "a")
    // r=2 (b): set={a, b}. len=2. count+=2. total=4 (chuoi "b", "ab")
    // Tong cong: 4 chuoi con don nhat: "a", "b", "a", "ab"
    string S2 = "abc";
    // Chuoi con don nhat: a, b, c, ab, bc, abc. Tong 6.
    // Phan tich: r=0 (1), r=1 (2), r=2 (3). Total: 1+2+3 = 6
    string S3 = "bbbbb";
    // Chuoi con don nhat: b, b, b, b, b. Tong 5.
    // Phan tich: r=0 (1). r=1 (trung, loai S[0]). len=1. total=2. r=2 (trung, loai
S[1]). len=1. total=3. ...
    cout << "S='" << S1 << "' | So luong chuoi con don nhat: " <<
count_distinct_substrings(S1) << endl; // 4
    cout << "S='" << S2 << "' | So luong chuoi con don nhat: " <<
count_distinct_substrings(S2) << endl; // 6
    cout << "S='" << S3 << "' | So luong chuoi con don nhat: " <<
count_distinct_substrings(S3) << endl; // 5
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
S='aab' | So luong chuoi con don nhat: 4
S='abc' | So luong chuoi con don nhat: 6
S='bbbbb' | So luong chuoi con don nhat: 5
```

2. Bài tập 2 (Phân tích Lặp lại): Cho xâu S. Tìm chuỗi con P dài nhất sao cho S chứa hai lần xuất hiện của P (có thể chồng lấn).

2.1. Code Python:

```
def longest_repeated_substring(S):
    # Tìm chuỗi con P dài nhất xuất hiện ít nhất hai lần trong xâu S.
    # Sử dụng phương pháp duyệt tất cả chuỗi con và tìm kiếm lại.
    # Độ phức tạp thời gian: O(N^3) (Tuy nhiên, có thể xem là O(N^2 * L) với L là
độ dài chuỗi con)
    N = len(S)
    # Duyệt độ dài (length) từ lớn nhất (N-1) xuống 1,
    # để khi tìm được kết quả, ta có thể dừng ngay.
    for length in range(N - 1, 0, -1):
        # Duyệt vị trí bắt đầu (i) của chuỗi con P
        # Chuỗi con phải đủ dài length: i <= N - length
        for i in range(N - length + 1):
            # Chuỗi con P
            P = S[i : i + length]
            # Kiểm tra xem P có xuất hiện lại ở vị trí khác không.
            # Bắt đầu tìm kiếm từ vị trí i + 1 để đảm bảo đó là lần xuất hiện thứ
hai,
            # và chấp nhận sự chồng lấn (overlap).
            # S.find(P, start_index)
            # start_index = i + 1
            second_occurrence_pos = S.find(P, i + 1)
            if second_occurrence_pos != -1:
                # Nếu tìm thấy P ở vị trí khác, P chính là chuỗi con lặp lại dài
nhất
                # (vì ta đang duyệt từ độ dài lớn nhất xuống).
                return P
    # Nếu không tìm thấy chuỗi con nào có độ dài >= 2 lặp lại
    return ""
# ----------------- CHẠY THỬ -----------------
# 1. Trường hợp lặp lại và chồng lấn (overlap)
```

```
S1 = "banana"
# Chuỗi con lặp: "ana" (xuất hiện ở index 1 và 3)
print(f"S='{S1}' | LRS: '{longest_repeated_substring(S1)}'") # ana
# 2. Trường hợp lặp lại không chồng lấn
S2 = "ababa"
# Chuỗi con lặp: "aba" (xuất hiện ở index 0 và 2)
print(f"S='{S2}' | LRS: '{longest_repeated_substring(S2)}'") # aba
# 3. Trường hợp không lặp lại
S3 = "abcde"
print(f"S='{S3}' | LRS: '{longest_repeated_substring(S3)}'") # (rỗng)
# 4. Trường hợp phức tạp
S4 = "abccab"
# Chuỗi con lặp: "ab" (index 0 và 4). Length=2.
# "c" (index 2 và 3). Length=1.
# LRS: "ab"
print(f"S='{S4}' | LRS: '{longest_repeated_substring(S4)}'") # ab
```

=> Kết quả sau khi chạy chương trình:

```
S='banana' | LRS: 'ana'
S='ababa' | LRS: 'aba'
S='abcde' | LRS: ''
S='abccab' | LRS: 'ab'
```

2.2. Code C++:

```
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
#include <string>            // Thư viện xử lí xâu
using namespace std;         // Tiền tố std
string longest_repeated_substring(const string &S)
{
    // Tim chuoi con P dai nhat xuat hien it nhat hai lan trong xau S.
    // Su dung phuong phap duyet tat ca chuoi con va tim kiem lai.
    // Do phuc tap thoi gian: O(N^3) (Tuy nhien, co the xem la O(N^2 * L) voi L la
do dai chuoi con)
    int N = S.length();
    // Duyet do dai (length) tu lon nhat (N-1) xuong 1,
    // de khi tim duoc ket qua, ta co the dung ngay.
    for (int length = N - 1; length > 0; length--)
    {
        // Duyet vi tri bat dau (i) cua chuoi con P
        // Chuoi con phai du dai length: i <= N - length
        for (int i = 0; i <= N - length; i++)
        {
            // Chuoi con P
            string P = S.substr(i, length);
            // Kiem tra xem P co xuat hien lai o vi tri khac khong.
            // Bat dau tim kiem tu vi tri i + 1 de dam bao do la lan xuat hien thu
hai,
            // va chap nhan su chong lan (overlap).
            // S.find(P, start_index)
            // start_index = i + 1
            size_t second_occurrence_pos = S.find(P, i + 1);
            if (second_occurrence_pos != string::npos)
            {
                // Neu tim thay P o vi tri khac, P chinh la chuoi con lap lai dai
nhat
                // (vi ta dang duyet tu do dai lon nhat xuong).
                return P;
            }
```

```
        }
    }
    // Neu khong tim thay chuoi con nao co do dai >= 2 lap lai
    return "";
}
int main()
{
    // ----------------- CHAY THU -----------------
    // 1. Truong hop lap lai va chong lan (overlap)
    string S1 = "banana";
    // Chuoi con lap: "ana" (xuat hien o index 1 va 3)
    cout << "S='" << S1 << "' | LRS: '" << longest_repeated_substring(S1) << "'" <<
endl; // ana
    // 2. Truong hop lap lai khong chong lan
    string S2 = "ababa";
    // Chuoi con lap: "aba" (xuat hien o index 0 va 2)
    cout << "S='" << S2 << "' | LRS: '" << longest_repeated_substring(S2) << "'" <<
endl; // aba
    // 3. Truong hop khong lap lai
    string S3 = "abcde";
    cout << "S='" << S3 << "' | LRS: '" << longest_repeated_substring(S3) << "'" <<
endl; // (rong)
    // 4. Truong hop phuc tap
    string S4 = "abccab";
    // Chuoi con lap: "ab" (index 0 va 4). Length=2.
    // "c" (index 2 va 3). Length=1.
    // LRS: "ab"
    cout << "S='" << S4 << "' | LRS: '" << longest_repeated_substring(S4) << "'" <<
endl; // ab
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
S='banana' | LRS: 'ana'
S='ababa' | LRS: 'aba'
S='abcde' | LRS: ''
S='abccab' | LRS: 'ab'
```

CHỦ ĐỀ 6. XỬ LÝ XÂU NÂNG CAO

BÀI 16. SO KHỚP XÂU CƠ BẢN

I. Mục Tiêu:

- **Kiến thức:** HS hiểu rõ bài toán So khớp Xâu (tìm kiếm mẫu P trong văn bản T). Nắm vững thuật toán vét cạn (**Brute Force**) và sự cần thiết của các thuật toán tối ưu.
- **Kỹ năng:** HS cài đặt được thuật toán **Brute Force** để tìm kiếm vị trí đầu tiên hoặc tất cả các vị trí xuất hiện của mẫu.
- **Tư duy:** HS phân tích được độ phức tạp thời gian của thuật toán vét cạn (O(N. M)) và nhận ra các trường hợp xấu nhất.

II. Phân Bổ Thời Gian (90 Phút):

Hoạt Động	Nội Dung	Thời gian (Phút)
Ôn tập	Ôn tập Chuỗi con Lặp lại dài nhất	10
Lý thuyết 1	Bài toán So khớp Xâu, Thuật toán Vét cạn (Brute Force)	15
Thực hành 1	Bài 1. Cài đặt Vét cạn tìm vị trí đầu tiên	25
Lý thuyết 2	Giới thiệu về Thuật toán Tối ưu (Rabin-Karp)	15
Thực hành 2	Bài 2. Cài đặt Vét cạn tìm tất cả vị trí	20
Tổng kết	Bài tập về nhà, Củng cố	5
Tổng cộng		**90**

III. Lý Thuyết 1: Thuật toán Vét cạn (Brute Force) (15 phút)

1. Bài toán So khớp Xâu (String Matching):

- **Input:**
  + **Văn bản T** (Text) có độ dài N.
  + **Mẫu P** (Pattern) có độ dài M.
- **Output:** Vị trí (chỉ mục) đầu tiên (hoặc tất cả các vị trí) mà mẫu P xuất hiện trong văn bản T.

2. Thuật toán Vét cạn (Brute Force Algorithm):

Đây là thuật toán đơn giản nhất:

- **Bước 1.** Duyệt qua tất cả các vị trí bắt đầu có thể có **i** trong văn bản **T** (từ **i = 0** đến **N - M**).
- **Bước 2.** Tại mỗi vị trí **i**, so sánh từng ký tự của mẫu **P** với từng ký tự tương ứng của **T** (tức là so sánh **P[j] với T[i+j]**).
- **Bước 3.** Nếu tất cả **M** ký tự đều khớp, ta đã tìm thấy một sự xuất hiện.

3. Độ phức tạp Thời gian:

Tối đa: O(N . M). Trường hợp xấu nhất xảy ra khi mẫu P và T khớp nhau gần hết ở mọi vị trí, ví dụ: T = "aaaa...ab, P = "aaa...b".

IV. Thực Hành 1: Cài đặt Vét cạn (25 phút):

1. Bài 1. Tìm vị trí xuất hiện đầu tiên của Mẫu P trong Văn bản T

1.1. Code Python:

```python
def brute_force_match_first(T, P):
    N = len(T)
    M = len(P)
    if M == 0:
        return 0    # Mẫu rỗng luôn khớp tại index 0
    if N < M:
        return -1   # T không đủ dài
    # Vòng lặp ngoài: Duyệt vị trí bắt đầu i trong T
    # i chạy từ 0 đến N - M
    for i in range(N - M + 1):
        # Vòng lặp trong: So sánh ký tự
        j = 0   # Chỉ mục của mẫu P
        # So sánh từng ký tự của P với T[i+j]
        while j < M and T[i + j] == P[j]:
            j += 1
```

```python
        # Kiểm tra sự xuất hiện
        if j == M:
            # Đã khớp toàn bộ M ký tự
            return i    # Trả về vị trí bắt đầu và kết thúc hàm
    # Lệnh return -1 PHẢI NẰM NGOÀI vòng lặp for,
    # chỉ thực hiện khi đã duyệt hết tất cả các vị trí i mà không tìm thấy.
    return -1
# --- Chạy thử ---
T = "ABABAABACDABABC"
P = "ABABC"
# P bắt đầu ở index 10: T[10]='A', T[11]='B', T[12]='A', T[13]='B', T[14]='C'
print(f"Vi tri dau tien: {brute_force_match_first(T, P)}")  # Kết quả đúng: 10
```

=> Kết quả sau khi chạy chương trình:

```
Vi tri dau tien: 10
```

- Giải thích Code Python:

+ **Vòng lặp i:** Duyệt từ 0 đến N-M (N - M + 1 trong range).

+ **Vòng lặp j:** So sánh ký tự. Nếu j == M, tức là tất cả M ký tự đã khớp.

1.2. Code C++:

```cpp
#include <iostream>         // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>    // Thư viện tổng hợp
#include <string>           // Thư viện xử lí xâu
using namespace std;        // Tiền tố std
int brute_force_match_first(const string& T, const string& P)
{
    int N = T.length();
    int M = P.length();
    if (M == 0)
        return 0;    // Mau rong luon khop tai index 0
    if (N < M)
        return -1;   // T khong du dai
    // Vong lap ngoai: Duyet vi tri bat dau i trong T
    // i chay tu 0 den N - M
    for (int i = 0; i <= N - M; i++)
    {
        // Vong lap trong: So sanh ky tu
        int j = 0;   // Chi muc cua mau P
        // So sanh tung ky tu cua P voi T[i+j]
        while (j < M && T[i + j] == P[j])
        {
            j++;
        }
        // Kiem tra su xuat hien
        if (j == M)
        {
            // Da khop toan bo M ky tu
            return i;    // Tra ve vi tri bat dau va ket thuc ham
        }
    }
    // Lenh return -1 PHAI NAM NGOAI vong lap for,
    // chi thuc hien khi da duyet het tat ca cac vi tri i ma khong tim thay.
    return -1;
}
// --- Chay thu ---
int main()
{
    string T = "ABABAABACDABABC";
```

```
    string P = "ABABC";
    // P bat dau o index 10: T[10]='A', T[11]='B', T[12]='A', T[13]='B', T[14]='C'
    cout << "Vi tri dau tien: " << brute_force_match_first(T, P) << endl;   // Ket
qua dung: 10
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
Vi tri dau tien: 10
```

- Giải thích Code C++:

+ Vòng lặp i: Điều kiện dừng là i <= N - M.

+ **Vòng lặp j:** Logic so sánh tương tự Python. C++ yêu cầu cẩn thận với kiểu dữ liệu **int** và **string::size_type**.

V. Lý thuyết 2: Giới thiệu về Thuật toán Tối ưu (15 phút):

1. Hạn chế của Vét cạn:

Trong trường hợp xấu nhất O(N . M), thuật toán Brute Force trở nên quá chậm khi xử lý các tập dữ liệu lớn (ví dụ: N, M $\leq 10^5$).

2. Nhu cầu về các thuật toán nhanh hơn:

Các thuật toán tối ưu hóa tìm kiếm dựa trên các ý tưởng sau:

- **Tiền xử lý Mẫu P:** (Ví dụ: KMP - Knuth-Morris-Pratt) Xây dựng bảng tiền tố để biết nên bỏ qua bao nhiêu ký tự khi có sự không khớp.

- **Sử dụng Hash:** (Ví dụ: Rabin-Karp) Sử dụng hàm băm (hash function) để so sánh các chuỗi con của T với P một cách nhanh chóng.

- **Duyệt từ Phải sang Trái:** (Ví dụ: Boyer-Moore) Bắt đầu so sánh từ cuối mẫu P và sử dụng quy tắc nhảy để bỏ qua các đoạn lớn của T.

3. Tập trung: Thuật toán Rabin-Karp

- **Ý tưởng:** So sánh giá trị băm (hash value) của mẫu P với giá trị băm của chuỗi con T[i..i+M-1].

- **Rolling Hash:** Tính toán giá trị băm của chuỗi con tiếp theo T[i+1..i+M] từ giá trị băm của T[i..i+M-1] trong thời gian O(1), giúp giảm tổng độ phức tạp.

- **Độ phức tạp:** Trung bình O(N+M), Trường hợp xấu nhất O(N . M) (do xung đột băm).

VI. Thực Hành 2: Tìm tất cả vị trí (20 phút)

1. Bài 2. Cài đặt Vét cạn tìm tất cả các vị trí xuất hiện

- **Yêu cầu:** Sửa đổi hàm Brute Force để tìm và trả về một danh sách (vector) chứa tất cả các chỉ mục bắt đầu của mẫu P trong T.

1.1. Code Python:

```
def brute_force_match_all(T, P):
    N = len(T)
    M = len(P)
    results = []        # Khởi tạo danh sách kết quả
    if M == 0:
        return results  # Mẫu rỗng không có vị trí hợp lệ trong bối cảnh này, hoặc
trả về []
    if N < M:
        return results
    # Vòng lặp ngoài: Duyệt vị trí bắt đầu i trong T
    for i in range(N - M + 1):
        # Vòng lặp trong: So sánh ký tự
        j = 0   # Chỉ mục của mẫu P
```

```
        # So sánh từng ký tự của P với T[i+j]
        while j < M and T[i + j] == P[j]:
            j += 1
        # Kiểm tra sự xuất hiện: KHÔNG ĐƯỢC đặt lệnh này trong vòng lặp while
        if j == M:
            # Đã khớp toàn bộ M ký tự
            results.append(i)   # Thêm vị trí vào danh sách
            # KHÔNG ĐƯỢC return ở đây, vì ta muốn tìm TẤT CẢ
    # Lệnh return results PHẢI NẰM NGOÀI vòng lặp for,
    # chỉ thực hiện khi đã duyệt hết tất cả các vị trí i.
    return results
# --- Chạy thử ---
T = "abababa"
P = "aba"
# P xuất hiện ở index 0, 2, và 4.
print(f"Tat ca vi tri: {brute_force_match_all(T, P)}")      # Kết quả đúng: [0, 2,
4]
T2 = "ABABAABACDABABC"
P2 = "ABABC"
print(f"Tat ca vi tri: {brute_force_match_all(T2, P2)}")    # Kết quả đúng: [10]
```

=> Kết quả sau khi chạy chương trình:

```
Tat ca vi tri: [0, 2, 4]
Tat ca vi tri: [10]
```

- Giải thích Code Python:

Sự khác biệt: Thay vì **return i** ngay sau khi tìm thấy, ta thêm **results.append(i)** và tiếp tục vòng lặp **for i** để tìm các sự xuất hiện tiếp theo.

1.2. Code C++:

```
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
#include <string>            // Thư viện xử lí xâu
using namespace std;         // Tiền tố std
vector<int> brute_force_match_all(const string& T, const string& P)
{
    int N = T.length();
    int M = P.length();
    vector<int> results;     // Khoi tao danh sach ket qua
    if (M == 0)
        return results;      // Mau rong khong co vi tri hop le trong boi canh nay,
hoac tra ve []
    if (N < M)
        return results;
    // Vong lap ngoai: Duyet vi tri bat dau i trong T
    for (int i = 0; i <= N - M; i++)
    {
        // Vong lap trong: So sanh ky tu
        int j = 0;  // Chi muc cua mau P
        // So sanh tung ky tu cua P voi T[i+j]
        while (j < M && T[i + j] == P[j])
        {
            j++;
        }
        // Kiem tra su xuat hien: KHONG DUOC dat lenh nay trong vong lap while
        if (j == M)
        {
            // Da khop toan bo M ky tu
            results.push_back(i);   // Them vi tri vao danh sach
```

```
            // KHONG DUOC return o day, vi ta muon tim TAT CA
        }
    }
    // Lenh return results PHAI NAM NGOAI vong lap for,
    // chi thuc hien khi da duyet het tat ca cac vi tri i.
    return results;
}
// --- Chay thu ---
int main()
{
    string T = "abababa";
    string P = "aba";
    // P xuat hien o index 0, 2, va 4.
    vector<int> res1 = brute_force_match_all(T, P);
    cout << "Tat ca vi tri: ";
    for (int x : res1) cout << x << " ";
    cout << endl;
    string T2 = "ABABAABACDABABC";
    string P2 = "ABABC";
    vector<int> res2 = brute_force_match_all(T2, P2);
    cout << "Tat ca vi tri: ";
    for (int x : res2) cout << x << " ";
    cout << endl;
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
Tat ca vi tri: 0 2 4
Tat ca vi tri: 10
```

VII. Bài Tập Về Nhà & Tổng Kết (5 phút):

1. Bài tập 1 (Kiểm tra Chồng lấn): Viết lại hàm **brute_force_match_all** sao cho nó chỉ tìm kiếm các sự xuất hiện **không chồng lấn** (non-overlapping). *(Gợi ý: Nếu tìm thấy ở vị trí i, thì lần tìm kiếm tiếp theo phải bắt đầu từ i + M).*

1.1. Code Python:

```python
def brute_force_match_non_overlapping(T, P):
    # Tìm tất cả các vị trí xuất hiện KHÔNG CHỒNG LẤN của mẫu P trong văn bản T.
    # Sử dụng thuật toán vét cạn với việc nhảy chỉ mục.
    # Độ phức tạp: O(N * M)
    N = len(T)
    M = len(P)
    results = []
    if M == 0:
        # Mẫu rỗng. Theo quy ước, có thể trả về 0 hoặc danh sách rỗng.
        # Trong bối cảnh tìm kiếm thực tế, ta thường trả về danh sách rỗng.
        return results
    if N < M:
        return results
    # Tự quản lý chỉ mục bắt đầu (i)
    i = 0 # Vị trí bắt đầu tìm kiếm hiện tại
    # Vòng lặp ngoài: Tiếp tục tìm kiếm cho đến khi hết T
    while i <= N - M:
        # Vòng lặp trong: So sánh ký tự tại vị trí i
        j = 0 # Chỉ mục của mẫu P
        # So sánh từng ký tự của P với T[i+j]
        while j < M and T[i + j] == P[j]:
            j += 1
        # Kiểm tra sự xuất hiện
```

```
        if j == M:
            # 1. TÌM THẤY SỰ KHỚP
            results.append(i)
            # 2. ĐIỀU CHỈNH CHỈ MỤC (Key step for non-overlapping)
            # Bắt đầu tìm kiếm tiếp theo ngay sau vị trí kết thúc của mẫu P
            i += M # Bắt đầu tìm kiếm từ i + M
        else:
            # KHÔNG TÌM THẤY tại vị trí i
            # Chuyển sang vị trí tiếp theo (i + 1)
            i += 1
    return results
# ----------------- CHẠY THỬ -----------------
# Ví dụ cho thấy sự khác biệt:
T1 = "abababa"
P1 = "aba"
# Kết quả (chồng lấn): [0, 2, 4]
# Kết quả (KHÔNG chồng lấn):
# - Tìm thấy tại 0. Nhảy đến 0 + 3 = 3.
# - Từ 3, tìm thấy tại 4. Nhảy đến 4 + 3 = 7. Hết xâu.
# -> [0, 4]
print(f"T='{T1}', P='{P1}'")
print(f"Chong lan (kiem tra): {[i for i in range(len(T1)) if T1.startswith(P1,
i)]}") # [0, 2, 4]
print(f"KHONG CHONG LAN: {brute_force_match_non_overlapping(T1, P1)}") # [0, 4]
print("-" * 30)
T2 = "banana_banana_banana"
P2 = "banana"
# Kết quả (KHÔNG chồng lấn): [0, 7, 14]
print(f"T='{T2}', P='{P2}'")
print(f"KHONG CHONG LAN: {brute_force_match_non_overlapping(T2, P2)}") # [0, 7, 14]
```

=> Kết quả sau khi chạy chương trình:

```
T='abababa', P='aba'
Chong lan (kiem tra): [0, 2, 4]
KHONG CHONG LAN: [0, 4]
------------------------------
T='banana_banana_banana', P='banana'
KHONG CHONG LAN: [0, 7, 14]
```

1.2. Code C++:

```
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
#include <string>            // Thư viện xử lí xâu
using namespace std;         // Tiền tố std
vector<int> brute_force_match_non_overlapping(const string& T, const string& P)
{
    // Tim tat ca cac vi tri xuat hien KHONG CHONG LAN cua mau P trong van ban T.
    // Su dung thuat toan vet can voi viec nhay chi muc.
    // Do phuc tap: O(N * M)
    int N = T.length();
    int M = P.length();
    vector<int> results;
    if (M == 0)
    {
        // Mau rong. Theo quy uoc, co the tra ve 0 hoac danh sach rong.
        // Trong boi canh tim kiem thuc te, ta thuong tra ve danh sach rong.
        return results;
    }
    if (N < M)
```

```
    {
        return results;
    }
    // Tu quan ly chi muc bat dau (i)
    int i = 0; // Vi tri bat dau tim kiem hien tai
    // Vong lap ngoai: Tiep tuc tim kiem cho den khi het T
    while (i <= N - M)
    {
        // Vong lap trong: So sanh ky tu tai vi tri i
        int j = 0; // Chi muc cua mau P
        // So sanh tung ky tu cua P voi T[i+j]
        while (j < M && T[i + j] == P[j])
        {
            j++;
        }
        // Kiem tra su xuat hien
        if (j == M)
        {
            // 1. TIM THAY SU KHOP
            results.push_back(i);
            // 2. DIEU CHINH CHI MUC (Key step for non-overlapping)
            // Bat dau tim kiem tiep theo ngay sau vi tri ket thuc cua mau P
            i += M; // Bat dau tim kiem tu i + M
        }
        else
        {
            // KHONG TIM THAY tai vi tri i
            // Chuyen sang vi tri tiep theo (i + 1)
            i += 1;
        }
    }
    return results;
}
// ----------------- CHAY THU -----------------
int main()
{
    string T1 = "abababa";
    string P1 = "aba";
    // Ket qua (KHONG chong lan)
    vector<int> r1 = brute_force_match_non_overlapping(T1, P1);
    cout << "T='" << T1 << "', P='" << P1 << "'\n";
    cout << "KHONG CHONG LAN: ";
    for (int x : r1) cout << x << " ";
    cout << "\n";
    cout << string(30, '-') << "\n";
    string T2 = "banana_banana_banana";
    string P2 = "banana";
    vector<int> r2 = brute_force_match_non_overlapping(T2, P2);
    cout << "T='" << T2 << "', P='" << P2 << "'\n";
    cout << "KHONG CHONG LAN: ";
    for (int x : r2) cout << x << " ";
    cout << "\n";
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
T='abababa', P='aba'
KHONG CHONG LAN: 0 4
------------------------------
T='banana_banana_banana', P='banana'
KHONG CHONG LAN: 0 7 14
```

2. Bài tập 2 (Rabin-Karp Cơ bản): Thử nghiệm xây dựng hàm băm cơ bản (polynomial hashing) cho một chuỗi con và viết hàm so sánh băm O(1) cho hai chuỗi con cùng độ dài. *(Đây là bước chuẩn bị cho buổi học chuyên sâu về Rabin-Karp).*

2.1. Code Python:

```python
# Chọn các hằng số
P = 31      # Cơ số (một số nguyên tố nhỏ)
M = 10**9 + 7 # Modulo (một số nguyên tố lớn)
def char_to_int(char):
    # Chuyển ký tự thành giá trị số (ví dụ: 'a' -> 1, 'b' -> 2, ...)
    # Ta giả định xâu chỉ chứa ký tự chữ thường Latin
    return ord(char) - ord('a') + 1
def compute_initial_hash(S, length):
    # Bước 1: Tính giá trị băm đa thức cho chuỗi con ban đầu S[0:length]
    current_hash = 0
    # Tính P^length để dùng cho việc loại bỏ ký tự sau này
    P_power = 1
    # Duyệt từ trái sang phải
    for i in range(length):
        # Hash = (Hash * P + Giá trị số của ký tự) mod M
        char_val = char_to_int(S[i])
        current_hash = (current_hash * P + char_val) % M
        # Cập nhật P^length
        if i < length - 1:
            P_power = (P_power * P) % M
    return current_hash, P_power
def compare_hash_O1(old_hash, old_char, new_char, P_power, M):
    # Bước 2: Tính hash mới từ hash cũ trong thời gian O(1) (Rolling Hash)
    # Tham số:
    # old_hash (int): Hash của S[i : i+L]
    # old_char (char): Ký tự bị loại bỏ (S[i])
    # new_char (char): Ký tự được thêm vào (S[i+L])
    # P_power (int): P^(L-1) mod M, cần để loại bỏ ký tự đầu tiên.
    # 1. Loại bỏ ký tự đầu tiên (S[i])
    # Giá trị của S[i] trong tổng là S[i] * P^(L-1)
    old_char_val = char_to_int(old_char)
    # Trừ đi phần đóng góp của ký tự cũ: Hash = Hash - (old_char_val * P^(L-1))
    # Phép trừ cần cẩn thận với modulo (thêm M để đảm bảo kết quả dương)
    new_hash = (old_hash - (old_char_val * P_power) % M + M) % M
    # 2. Dịch chuyển phần còn lại sang trái (nhân với P)
    # new_hash = (Hash còn lại) * P
    new_hash = (new_hash * P) % M
    # 3. Thêm ký tự mới (S[i+L])
    # new_hash = new_hash + new_char_val
    new_hash = (new_hash + char_to_int(new_char)) % M
    return new_hash
# ----------------- CHẠY THỬ -----------------
S = "abracadabra"
LENGTH = 5 # Độ dài chuỗi con cần kiểm tra
# 1. Tính hash ban đầu cho "abrac" (S[0:5])
initial_hash, P_power_L_minus_1 = compute_initial_hash(S, LENGTH)
print(f"Xau goc: {S}")
```

```
print(f"P^4 mod M (dung de loai bo): {P_power_L_minus_1}")
print(f"Hash cua '{S[0:LENGTH]}': {initial_hash}")
# 2. Tính hash cho chuỗi con tiếp theo "braca" (S[1:6])
# Loại bỏ: S[0]='a', Thêm vào: S[5]='a'
next_hash = compare_hash_O1(
    initial_hash,
    S[0],      # Ký tự cũ bị loại bỏ
    S[LENGTH], # Ký tự mới được thêm vào
    P_power_L_minus_1,
    M
)
print(f"\nHash cua '{S[1:1+LENGTH]}': {next_hash}")
# 3. Tính hash cho chuỗi con tiếp theo "racad" (S[2:7])
# Loại bỏ: S[1]='b', Thêm vào: S[6]='d'
next_hash_2 = compare_hash_O1(
    next_hash,
    S[1],      # Ký tự cũ bị loại bỏ
    S[LENGTH + 1], # Ký tự mới được thêm vào
    P_power_L_minus_1,
    M
)
print(f"Hash cua '{S[2:2+LENGTH]}': {next_hash_2}")
# 4. Kiểm tra sự trùng khớp
# Hash của 'abrac' (index 0) và 'cadab' (index 4)
# Hash của 'cadab'
hash_cadab = compare_hash_O1(
    next_hash_2,
    S[2],      # Ký tự cũ bị loại bỏ ('r')
    S[LENGTH + 2], # Ký tự mới được thêm vào ('a')
    P_power_L_minus_1,
    M
)
print(f"\nHash cua '{S[4:4+LENGTH]}': {hash_cadab}")
print(f"Hash '{S[0:LENGTH]}' == Hash '{S[4:4+LENGTH]}'? {initial_hash ==
hash_cadab}")
# Kết quả: True (abrac != cadab, nhưng hash trùng -> Xung đột hash!)
# Điều này cho thấy vai trò của số nguyên tố lớn M và kiểm tra ký tự cuối cùng.
```

=> Kết quả sau khi chạy chương trình:

```
Xau goc: abracadabra
P^4 mod M (dung de loai bo): 923521
Hash cua 'abrac': 1000435

Hash cua 'braca': 2384335
Hash cua 'racad': 16656087

Hash cua 'cadab': 1013980
Hash 'abrac' == Hash 'cadab'? False
```

2.2. Code C++:

```
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
#include <string>            // Thư viện xử lí xâu
using namespace std;         // Tiền tố std
// Chon cac hang so
const long long P = 31;           // Co so (mot so nguyen to nho)
const long long M = 1000000007;   // Modulo (mot so nguyen to lon)
```

```
int char_to_int(char c)
{
    // Chuyen ky tu thanh gia tri so (vi du: 'a' -> 1, 'b' -> 2, ...)
    // Ta gia dinh xau chi chua ky tu chu thuong Latin
    return c - 'a' + 1;
}
pair<long long, long long> compute_initial_hash(const string& S, int length)
{
    // Buoc 1: Tinh gia tri bam da thuc cho chuoi con ban dau S[0:length]
    long long current_hash = 0;
    // Tinh P^length de dung cho viec loai bo ky tu sau nay
    long long P_power = 1;
    // Duyet tu trai sang phai
    for (int i = 0; i < length; i++)
    {
        // Hash = (Hash * P + Gia tri so cua ky tu) mod M
        int char_val = char_to_int(S[i]);
        current_hash = (current_hash * P + char_val) % M;
        // Cap nhat P^length
        if (i < length - 1)
        {
            P_power = (P_power * P) % M;
        }
    }
    return {current_hash, P_power};
}
long long compare_hash_O1(long long old_hash, char old_char, char new_char,
                          long long P_power, long long M)
{
    // Buoc 2: Tinh hash moi tu hash cu trong thoi gian O(1) (Rolling Hash)
    // Tham so:
    // old_hash (int): Hash cua S[i : i+L]
    // old_char (char): Ky tu bi loai bo (S[i])
    // new_char (char): Ky tu duoc them vao (S[i+L])
    // P_power (int): P^(L-1) mod M, can de loai bo ky tu dau tien.
    // 1. Loai bo ky tu dau tien (S[i])
    // Gia tri cua S[i] trong tong la S[i] * P^(L-1)
    long long old_char_val = char_to_int(old_char);
    // Tru di phan dong gop cua ky tu cu:
    // Hash = Hash - (old_char_val * P^(L-1))
    // Phep tru can can than voi modulo (them M de dam bao ket qua duong)
    long long new_hash = (old_hash - (old_char_val * P_power) % M + M) % M;
    // 2. Dich chuyen phan con lai sang trai (nhan voi P)
    // new_hash = (Hash con lai) * P
    new_hash = (new_hash * P) % M;
    // 3. Them ky tu moi (S[i+L])
    // new_hash = new_hash + new_char_val
    new_hash = (new_hash + char_to_int(new_char)) % M;
    return new_hash;
}
// ----------------- CHAY THU -----------------
int main()
{
    string S = "abracadabra";
    int LENGTH = 5; // Do dai chuoi con can kiem tra
    // 1. Tinh hash ban dau cho "abrac" (S[0:5])
    auto initial = compute_initial_hash(S, LENGTH);
    long long initial_hash = initial.first;
```

```
    long long P_power_L_minus_1 = initial.second;
    cout << "Xau goc: " << S << "\n";
    cout << "P^4 mod M (dung de loai bo): " << P_power_L_minus_1 << "\n";
    cout << "Hash cua '" << S.substr(0, LENGTH) << "': " << initial_hash << "\n";
    // 2. Tinh hash cho chuoi con tiep theo "braca" (S[1:6])
    // Loai bo: S[0]='a', Them vao: S[5]='a'
    long long next_hash = compare_hash_O1(
        initial_hash,
        S[0],          // Ky tu cu bi loai bo
        S[LENGTH],     // Ky tu moi duoc them vao
        P_power_L_minus_1,
        M
    );
    cout << "\nHash cua '" << S.substr(1, LENGTH) << "': " << next_hash << "\n";
    // 3. Tinh hash cho chuoi con tiep theo "racad" (S[2:7])
    long long next_hash_2 = compare_hash_O1(
        next_hash,
        S[1],               // Ky tu cu bi loai bo
        S[LENGTH + 1],      // Ky tu moi
        P_power_L_minus_1,
        M
    );
    cout << "Hash cua '" << S.substr(2, LENGTH) << "': " << next_hash_2 << "\n";
    // 4. Kiem tra su trung khop
    // Hash cua 'abrac' (index 0) va 'cadab' (index 4)
    long long hash_cadab = compare_hash_O1(
        next_hash_2,
        S[2],               // Ky tu cu bi loai bo ('r')
        S[LENGTH + 2],      // Ky tu moi ('a')
        P_power_L_minus_1,
        M
    );
    cout << "\nHash cua '" << S.substr(4, LENGTH) << "': " << hash_cadab << "\n";
    cout << "Hash '" << S.substr(0, LENGTH) << "' == Hash '" << S.substr(4, LENGTH)
         << "'? " << (initial_hash == hash_cadab ? "True" : "False") << "\n";
    // Ket qua: True (abrac != cadab, nhung hash trung -> Xung dot hash!)
    // Dieu nay cho thay vai tro cua so nguyen to lon M va kiem tra ky tu cuoi
cung.
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
Xau goc: abracadabra
P^4 mod M (dung de loai bo): 923521
Hash cua 'abrac': 1000435

Hash cua 'braca': 2384335
Hash cua 'racad': 16656087

Hash cua 'cadab': 1013980
Hash 'abrac' == Hash 'cadab'? False
```

BÀI 17. THUẬT TOÁN KMP (KNUTH – MORRIS – PRATT)

I. Mục Tiêu:

- **Kiến thức:** HS hiểu rõ hạn chế của Brute Force và tại sao KMP lại hiệu quả hơn. Nắm vững khái niệm về **Bảng tiền tố** (Prefix Function/LPS Array) và vai trò của nó.
- **Kỹ năng:** HS cài đặt được hàm xây dựng Bảng tiền tố O(M) và thuật toán tìm kiếm KMP O(N).
- **Tư duy:** HS phân tích được độ phức tạp thời gian tối ưu của KMP (O(N+M)) và sự khác biệt giữa thuật toán này với các thuật toán O(N . M).

II. Phân Bổ Thời Gian (90 Phút):

Hoạt Động	Nội Dung	Thời gian (Phút)
Ôn tập	Ôn tập Vét cạn, Giới thiệu về hiệu quả của KMP	10
Lý thuyết 1	Khái niệm Bảng tiền tố (LPS Array)	15
Thực hành 1	Bài 1. Xây dựng Bảng tiền tố (KMP Preprocessing)	30
Lý thuyết 2	Cơ chế dịch chuyển của KMP (KMP Matching)	15
Thực hành 2	Bài 2. Cài đặt thuật toán KMP hoàn chỉnh	15
Tổng kết	Bài tập về nhà, Củng cố	5
Tổng cộng		**90**

III. Lý Thuyết 1: Bảng tiền tố (LPS Array) (15 phút)

1. Hạn chế của Brute Force:

Khi xảy ra sự không khớp (mismatch), thuật toán Brute Force phải dịch chuyển mẫu P sang phải **chỉ 1 vị trí** và bắt đầu so sánh lại từ đầu. Điều này lãng phí vì ta đã biết các ký tự đã khớp trước đó.

2. Khái niệm Bảng tiền tố (LPS Array):

- **LPS (Longest Proper Prefix which is also a Suffix):** Là độ dài của tiền tố **thích hợp** (proper prefix) dài nhất của P[0..i] mà tiền tố đó cũng là hậu tố của P[0..i].

  Tiền tố thích hợp: Mọi tiền tố trừ chính nó.

- **Mục đích:** Khi xảy ra không khớp tại P[j], bảng tiền tố cho ta biết nên dịch chuyển P sao cho tiền tố đã khớp của P (độ dài LPS[j-1]) được đặt ngay sau vị trí không khớp, tránh phải so sánh lại.
- **Ký hiệu:** Bảng π(Pi) hoặc lps.
- **Ví dụ:** P = "ababa"
  + P[0..0] = "a": LPS = 0
  + P[0..1] = "ab": LPS = 0
  + P[0..2] = "aba": LPS = 1 (Tiền tố a là hậu tố a)
  + P[0..3] = "abab": LPS = 2 (Tiền tố ab là hậu tố ab)
  + P[0..4] = "ababa": LPS = 3 (Tiền tố aba là hậu tố aba)
  + **Bảng LPS:** [0, 0, 1, 2, 3]

IV. Thực Hành 1: Xây dựng Bảng tiền tố (30 phút)

1. Bài 1. Hàm Xây dựng Bảng tiền tố (KMP Preprocessing)

Thuật toán xây dựng Bảng tiền tố sử dụng kỹ thuật tương tự như thuật toán KMP tìm kiếm, trong đó mẫu P được so khớp với chính nó.

1.1. Code Python:

```python
def compute_lps_array(P):
    M = len(P)
    lps = [0] * M   # Khởi tạo bảng LPS
    length = 0      # Độ dài của tiền tố dài nhất hiện tại (LPS[i-1])
    i = 1           # Chỉ mục hiện tại (bắt đầu từ 1)
    while i < M:
        if P[i] == P[length]:
            # Trường hợp 1: Khớp. Mở rộng tiền tố/hậu tố
            length += 1
            lps[i] = length
            i += 1
        else:
            # Trường hợp 2: Không khớp
            if length != 0:
                # Thử tiền tố/hậu tố ngắn hơn
                length = lps[length - 1]
                # KHÔNG tăng i (so sánh lại với P[i] hiện tại)
            else:
                # Không khớp và length = 0
                lps[i] = 0
                i += 1
    return lps
P1 = "ababa"
P2 = "aabaaac"
print(f"LPS('{P1}'): {compute_lps_array(P1)}")  # [0, 0, 1, 2, 3]
print(f"LPS('{P2}'): {compute_lps_array(P2)}")  # [0, 1, 0, 1, 2, 2, 3]
```

=> Kết quả sau khi chạy chương trình:

```
LPS('ababa'): [0, 0, 1, 2, 3]
LPS('aabaaac'): [0, 1, 0, 1, 2, 2, 0]
```

- Giải thích Code Python:

Độ phức tạp O(M): Mặc dù có vòng lặp **while** lồng trong **if/else**, tổng cộng **length** chỉ tăng **M** lần và giảm tối đa **M** lần.

1.2. Code C++:

```cpp
#include <iostream>         // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>    // Thư viện tổng hợp
#include <string>           // Thư viện xử lí xâu
using namespace std;        // Tiền tố std
vector<int> compute_lps_array(const string& P)
{
    int M = P.length();
    vector<int> lps(M, 0);   // Khoi tao bang LPS
    int length = 0;          // Do dai cua tien to dai nhat hien tai (LPS[i-1])
    int i = 1;               // Chi muc hien tai (bat dau tu 1)
    while (i < M)
    {
        if (P[i] == P[length])
        {
            // Truong hop 1: Khop. Mo rong tien to/hau to
            length += 1;
```

```
            lps[i] = length;
            i += 1;
        }
        else
        {
            // Truong hop 2: Khong khop
            if (length != 0)
            {
                // Thu tien to/hau to ngan hon
                length = lps[length - 1];
                // KHONG tang i (so sanh lai voi P[i] hien tai)
            }
            else
            {
                // Khong khop va length = 0
                lps[i] = 0;
                i += 1;
            }
        }
    }
    return lps;
}
// ----------------- CHAY THU -----------------
int main()
{
    string P1 = "ababa";
    string P2 = "aabaaac";
    vector<int> lps1 = compute_lps_array(P1);
    vector<int> lps2 = compute_lps_array(P2);
    cout << "LPS('" << P1 << "'): [";
    for (int i = 0; i < lps1.size(); i++)
    {
        cout << lps1[i];
        if (i + 1 < lps1.size()) cout << ", ";
    }
    cout << "]\n";
    cout << "LPS('" << P2 << "'): [";
    for (int i = 0; i < lps2.size(); i++)
    {
        cout << lps2[i];
        if (i + 1 < lps2.size()) cout << ", ";
    }
    cout << "]\n";
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
LPS('ababa'): [0, 0, 1, 2, 3]
LPS('aabaaac'): [0, 1, 0, 1, 2, 2, 0]
```

- Giải thích Code C++:

Vector: Sử dụng **vector<int>** để lưu trữ mảng LPS. Logic so sánh và dịch chuyển **length** hoàn toàn tương tự như Python.

V. Lý thuyết 2: Cơ chế dịch chuyển của KMP (15 phút)

- Cơ chế KMP Matching: Thuật toán KMP sử dụng Bảng tiền tố (lps) để hướng dẫn việc dịch chuyển mẫu P khi có sự không khớp xảy ra trong quá trình so sánh P với T.

- **Duy trì con trỏ:**
  + i: Duyệt qua Văn bản T (từ 0 đến N-1).
  + j: Duyệt qua Mẫu P (từ 0 đến M-1), cũng là độ dài của phần đã khớp.
- **Khớp (Match):** Nếu T[i] = P[j], tăng cả i và j.
- **Không Khớp (Mismatch) tại P[j]:**
  + **Nếu j ≠ 0:** Dịch chuyển mẫu P bằng cách đặt j = lps[j-1]. (Bắt đầu so sánh lại từ P[lps}[j-1]]).
  + **Nếu j = 0:** Không thể dịch chuyển P. Chỉ cần tăng i và tiếp tục so sánh P[0] với T[i+1].
- **Tìm thấy sự Khớp:** Nếu j = M, ta đã tìm thấy một sự xuất hiện tại vị trí i-j trong T.
  + Sau khi ghi nhận kết quả, ta lại dịch chuyển mẫu: j = lps[j-1], và tiếp tục tìm kiếm.
- **Độ phức tạp:** O(N) (vì con trỏ i chỉ tăng N lần, con trỏ j chỉ tăng tối đa N lần và giảm tối đa N lần).
- **Tổng thể KMP:** O(M) (tiền xử lý) + O(N) (tìm kiếm) = O(N+M).

VI. Thực Hành 2: Cài đặt KMP hoàn chỉnh (15 phút)

1. Bài 2. Hàm Tìm kiếm KMP hoàn chỉnh

Chúng ta sẽ kết hợp hàm **compute_lps_array** và **logic KMP matching.**

1.1. Code Python:

```python
# Hàm tìm kiếm mảng lps (Longest Prefix Suffix)
def compute_lps_array(P):
    M = len(P)
    lps = [0] * M   # Khởi tạo bảng LPS
    length = 0      # Độ dài của tiền tố dài nhất hiện tại (LPS[i-1])
    i = 1           # Chỉ mục hiện tại (bắt đầu từ 1)
    while i < M:
        if P[i] == P[length]:
            # Trường hợp 1: Khớp. Mở rộng tiền tố/hậu tố
            length += 1
            lps[i] = length
            i += 1
        else:
            # Trường hợp 2: Không khớp
            if length != 0:
                # Thử tiền tố/hậu tố ngắn hơn
                length = lps[length - 1]
                # KHÔNG tăng i (so sánh lại với P[i] hiện tại)
            else:
                # Không khớp và length = 0
                lps[i] = 0
                i += 1
    return lps
# Hàm tìm kiếm KMP
def kmp_search(T, P):
    N = len(T)
    M = len(P)
    results = []
    if M == 0:
        # Tùy theo yêu cầu, có thể trả về [0] nếu mẫu rỗng khớp tại đầu
        return results
```

```python
    # 1. Tiền xử lý: Xây dựng Bảng tiền tố LPS
    lps = compute_lps_array(P)
    # 2. Tìm kiếm KMP
    i = 0  # Chỉ mục cho T (text)
    j = 0  # Chỉ mục cho P (pattern)
    while i < N:
        # --- BƯỚC 1: Xử lý Khớp ---
        if T[i] == P[j]:
            i += 1
            j += 1
        # --- BƯỚC 2: Xử lý Tìm thấy Khớp ---
        if j == M:
            # Tìm thấy sự khớp. Vị trí bắt đầu là i - M (hoặc i - j)
            results.append(i - j)
            # Chuẩn bị cho lần tìm kiếm tiếp theo (Dịch chuyển P):
            # Thiết lập j về độ dài tiền tố/hậu tố đã khớp trước đó
            j = lps[j - 1]
        # --- BƯỚC 3: Xử lý Không Khớp ---
        # Kiểm tra nếu T[i] và P[j] không khớp (chỉ xảy ra khi j chưa đạt M)
        elif i < N and T[i] != P[j]:
            if j != 0:
                # Dịch chuyển P: Đặt j về giá trị LPS trước đó
                # i KHÔNG dịch chuyển, ta so sánh lại T[i] với P[j] mới
                j = lps[j - 1]
            else:
                # j=0 (không còn tiền tố/hậu tố nào để dịch chuyển)
                # Chỉ cần dịch chuyển i sang ký tự tiếp theo trong T
                i += 1
    # Trả về kết quả sau khi duyệt hết T
    return results
# --- Chạy thử ---
# (Cần chạy lại hàm compute_lps_array trước khi chạy đoạn này)
# P = "ABABCABA", lps = [0, 0, 1, 2, 0, 1, 2, 3]
T = "ABABDABACDABABCABA"
P = "ABABCABA"
print(f"KMP Result: {kmp_search(T, P)}")
# Kết quả đúng: [10] (Vì P bắt đầu tại T[10])
```

=> Kết quả sau khi chạy chương trình:

```
KMP Result: [10]
```

- Giải thích Code Python:

Thứ tự if: Rất quan trọng! Kiểm tra j == M (tìm thấy) phải nằm **sau** khi tăng i và j (bước 1), và **trước** khi kiểm tra T[i] != P[j] (bước 3).

1.2. Code C++:

```cpp
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
#include <string>            // Thư viện xử lí xâu
using namespace std;         // Tiền tố std
// Ham tinh mang LPS (Longest Prefix Suffix)
vector<int> compute_lps_array(const string &P)
{
    int M = P.length();
    vector<int> lps(M, 0);  // Khoi tao bang LPS
    int length = 0;         // Do dai cua tien to dai nhat hien tai (LPS[i-1])
    int i = 1;              // Chi muc hien tai (bat dau tu 1)
    while (i < M)
    {
```

```
        if (P[i] == P[length])
        {
            // Truong hop 1: Khop. Mo rong tien to/hau to
            length++;
            lps[i] = length;
            i++;
        }
        else
        {
            // Truong hop 2: Khong khop
            if (length != 0)
            {
                // Thu tien to/hau to ngan hon
                length = lps[length - 1];
                // KHONG tang i (so sanh lai voi P[i] hien tai)
            }
            else
            {
                // Khong khop va length = 0
                lps[i] = 0;
                i++;
            }
        }
    }
    return lps;
}
// Ham tim kiem KMP
vector<int> kmp_search(const string &T, const string &P)
{
    int N = T.length();
    int M = P.length();
    vector<int> results;
    if (M == 0)
    {
        // Tuy theo yeu cau, co the tra ve [0] neu mau rong khop tai dau
        return results;
    }
    // 1. Tien xu ly: Xay dung Bang tien to LPS
    vector<int> lps = compute_lps_array(P);
    // 2. Tim kiem KMP
    int i = 0;  // Chi muc cho T (text)
    int j = 0;  // Chi muc cho P (pattern)
    while (i < N)
    {
        // --- BUOC 1: Xu ly Khop ---
        if (T[i] == P[j])
        {
            i++;
            j++;
        }
        // --- BUOC 2: Xu ly Tim thay Khop ---
        if (j == M)
        {
            // Tim thay su khop. Vi tri bat dau la i - M (hoac i - j)
            results.push_back(i - j);
            // Chuan bi cho lan tim kiem tiep theo (Dich chuyen P):
            // Thiet lap j ve do dai tien to/hau to da khop truoc do
            j = lps[j - 1];
```

```
        }
        // --- BUOC 3: Xu ly Khong Khop ---
        // Kiem tra neu T[i] va P[j] khong khop (chi xay ra khi j chua dat M)
        else if (i < N && T[i] != P[j])
        {
            if (j != 0)
            {
                // Dich chuyen P: Dat j ve gia tri LPS truoc do
                // i KHONG dich chuyen, ta so sanh lai T[i] voi P[j] moi
                j = lps[j - 1];
            }
            else
            {
                // j = 0 (khong con tien to/hau to nao de dich chuyen)
                // Chi can dich chuyen i sang ky tu tiep theo trong T
                i++;
            }
        }
    }
    // Tra ve ket qua sau khi duyet het T
    return results;
}
int main()
{
    // --- Chay thu ---
    // (Can chay lai ham compute_lps_array truoc khi chay doan nay)
    // P = "ABABCABA", lps = [0, 0, 1, 2, 0, 1, 2, 3]
    string T = "ABABDABACDABABCABA";
    string P = "ABABCABA";
    vector<int> result = kmp_search(T, P);
    cout << "KMP Result: ";
    for (int pos : result)
    {
        cout << pos << " ";
    }
    cout << endl;
    // Ket qua dung: [10] (Vi P bat dau tai T[10])
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
KMP Result: 10
```

- Giải thích Code C++:

Xử lý Biên: Cần đảm bảo điều kiện **i < N** trong khối **else if** để tránh truy cập ngoài phạm vi (out-of-bounds access) của **T** khi vòng lặp while kết thúc.

VII. Bài Tập Về Nhà & Tổng Kết (5 phút)

1. Bài tập 1 (Đếm Lần Xuất hiện): Viết lại hàm **kmp_search** để chỉ đếm **tổng số lần** P xuất hiện trong T (không cần lưu trữ các vị trí).

1.1. Code Python:

```
# Hàm Xây dựng Bảng tiền tố (LPS Array)
def compute_lps_array(P):
    # Xây dựng mảng tiền tố dài nhất cũng là hậu tố (LPS Array)
    M = len(P)
    lps = [0] * M
    length = 0
    i = 1
```

```
    while i < M:
        if P[i] == P[length]:
            length += 1
            lps[i] = length
            i += 1
        else:
            if length != 0:
                length = lps[length - 1]
            else:
                lps[i] = 0
                i += 1
    return lps
# Hàm KMP Đếm Lần Xuất hiện
def kmp_count_occurrences(T, P):
    # Đếm tổng số lần xuất hiện (có thể chồng lấn) của mẫu P trong văn bản T
    # bằng thuật toán KMP.
    # Độ phức tạp: O(N + M)
    N = len(T)
    M = len(P)
    if M == 0:
        # Mẫu rỗng. Theo quy ước bài toán, ta thường không đếm hoặc trả về 0.
        return 0
    lps = compute_lps_array(P)
    count = 0  # Bộ đếm tổng số lần xuất hiện
    i = 0  # Chỉ mục cho T (text)
    j = 0  # Chỉ mục cho P (pattern / độ dài đã khớp)
    while i < N:
        # --- BƯỚC 1: Xử lý Khớp ---
        if T[i] == P[j]:
            i += 1
            j += 1
        # --- BƯỚC 2: Xử lý Tìm thấy Khớp ---
        if j == M:
            # TÌM THẤY: Tăng bộ đếm
            count += 1
            # Chuẩn bị cho lần tìm kiếm tiếp theo (dịch chuyển P):
            # Thiết lập j về độ dài tiền tố/hậu tố đã khớp trước đó
            # (Luôn cho phép chồng lấn trong KMP tiêu chuẩn)
            j = lps[j - 1]
        # --- BƯỚC 3: Xử lý Không Khớp ---
        elif i < N and T[i] != P[j]:
            if j != 0:
                # Dịch chuyển P bằng cách sử dụng giá trị LPS
                j = lps[j - 1]
            else:
                # j=0, dịch chuyển i
                i += 1
    return count
# ----------------- CHẠY THỬ -----------------
# 1. Trường hợp chồng lấn
T1 = "ababa"
P1 = "aba"
# Xuất hiện tại index 0, 2. (2 lần)
print(f"T='{T1}', P='{P1}' | So lan xuat hien: {kmp_count_occurrences(T1, P1)}") #
2
# 2. Trường hợp lặp lại nhiều lần
T2 = "aaaaabaaaaa"
P2 = "aaa"
```

```
# Xuất hiện tại index 0, 1, 2, 6, 7, 8. (6 lần)
print(f"T='{T2}', P='{P2}' | So lan xuat hien: {kmp_count_occurrences(T2, P2)}") #
6
# 3. Trường hợp không tìm thấy
T3 = "xyzabc"
P3 = "yz"
print(f"T='{T3}', P='{P3}' | So lan xuat hien: {kmp_count_occurrences(T3, P3)}") #
1
```

=> Kết quả sau khi chạy chương trình:

```
T='ababa', P='aba' | So lan xuat hien: 2
T='aaaaabaaaaa', P='aaa' | So lan xuat hien: 6
T='xyzabc', P='yz' | So lan xuat hien: 1
```

1.2. Code C++:

```
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
#include <string>            // Thư viện xử lí xâu
using namespace std;         // Tiền tố std
// Ham Xay dung Bang tien to (LPS Array)
vector<int> compute_lps_array(const string &P)
{
    // Xay dung mang tien to dai nhat cung la hau to (LPS Array)
    int M = P.length();
    vector<int> lps(M, 0);
    int length = 0;
    int i = 1;
    while (i < M)
    {
        if (P[i] == P[length])
        {
            length++;
            lps[i] = length;
            i++;
        }
        else
        {
            if (length != 0)
            {
                length = lps[length - 1];
            }
            else
            {
                lps[i] = 0;
                i++;
            }
        }
    }
    return lps;
}
// Ham KMP Dem Lan Xuat hien
int kmp_count_occurrences(const string &T, const string &P)
{
    // Dem tong so lan xuat hien (co the chong lan) cua mau P trong van ban T
    // bang thuat toan KMP.
    // Do phuc tap: O(N + M)
    int N = T.length();
    int M = P.length();
    if (M == 0)
```

```
    {
        // Mau rong. Theo quy uoc bai toan, ta thuong khong dem hoac tra ve 0.
        return 0;
    }
    vector<int> lps = compute_lps_array(P);
    int count = 0;  // Bo dem tong so lan xuat hien
    int i = 0;  // Chi muc cho T (text)
    int j = 0;  // Chi muc cho P (pattern / do dai da khop)
    while (i < N)
    {
        // --- BUOC 1: Xu ly Khop ---
        if (T[i] == P[j])
        {
            i++;
            j++;
        }
        // --- BUOC 2: Xu ly Tim thay Khop ---
        if (j == M)
        {
            // TIM THAY: Tang bo dem
            count++;
            // Chuan bi cho lan tim kiem tiep theo (dich chuyen P):
            // Thiet lap j ve do dai tien to/hau to da khop truoc do
            // (Luon cho phep chong lan trong KMP tieu chuan)
            j = lps[j - 1];
        }
        // --- BUOC 3: Xu ly Khong Khop ---
        else if (i < N && T[i] != P[j])
        {
            if (j != 0)
            {
                // Dich chuyen P bang cach su dung gia tri LPS
                j = lps[j - 1];
            }
            else
            {
                // j = 0, dich chuyen i
                i++;
            }
        }
    }
    return count;
}
int main()
{
    // ----------------- CHAY THU -----------------
    // 1. Truong hop chong lan
    string T1 = "ababa";
    string P1 = "aba";
    // Xuat hien tai index 0, 2. (2 lan)
    cout << "T='" << T1 << "', P='" << P1
         << "' | So lan xuat hien: "
         << kmp_count_occurrences(T1, P1) << endl;  // 2
    // 2. Truong hop lap lai nhieu lan
    string T2 = "aaaaabaaaaa";
    string P2 = "aaa";
    // Xuat hien tai index 0, 1, 2, 6, 7, 8. (6 lan)
    cout << "T='" << T2 << "', P='" << P2
```

```
        << "' | So lan xuat hien: "
        << kmp_count_occurrences(T2, P2) << endl;  // 6
    // 3. Truong hop khong tim thay
    string T3 = "xyzabc";
    string P3 = "yz";
    cout << "T='" << T3 << "', P='" << P3
        << "' | So lan xuat hien: "
        << kmp_count_occurrences(T3, P3) << endl;  // 1
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
T='ababa', P='aba' | So lan xuat hien: 2
T='aaaaabaaaaa', P='aaa' | So lan xuat hien: 6
T='xyzabc', P='yz' | So lan xuat hien: 1
```

2. Bài tập 2 (Prefixes và Suffixes): Sử dụng Bảng tiền tố (lps) để tìm **tất cả** các tiền tố của P mà cũng là hậu tố của P.

2.1. Code Python:

```
def compute_lps_array(P):
    # Hàm xây dựng mảng tiền tố dài nhất cũng là hậu tố (LPS Array)
    M = len(P)
    lps = [0] * M
    length = 0
    i = 1
    while i < M:
        if P[i] == P[length]:
            length += 1
            lps[i] = length
            i += 1
        else:
            if length != 0:
                length = lps[length - 1]
            else:
                lps[i] = 0
                i += 1
    return lps
def find_all_prefix_suffixes(P):
    # Sử dụng mảng LPS để tìm tất cả các tiền tố của P mà cũng là hậu tố của P.
    # Trả về danh sách các ĐỘ DÀI của các tiền tố/hậu tố này.
    M = len(P)
    if M == 0:
        return []
    # Bước 1: Tiền xử lý để có được mảng LPS
    lps = compute_lps_array(P)
    results = []
    # Bắt đầu với độ dài tiền tố/hậu tố dài nhất của toàn bộ xâu P.
    # LPS[M-1] là độ dài của tiền tố dài nhất của P[0..M-1] mà cũng là hậu tố của
nó.
    length = lps[M - 1]
    # Bước 2: Truy vết ngược (Backtracking)
    while length > 0:
        # Thêm độ dài hiện tại vào danh sách
        results.append(length)
        # Nhảy về độ dài tiền tố/hậu tố dài nhất của tiền tố/hậu tố hiện tại
        # Ví dụ: Nếu P = "ababa" (M=5), LPS[4]=3 ("aba").
        # Tiếp theo, ta kiểm tra LPS[3-1] = LPS[2] = 1 ("a").
        length = lps[length - 1]
```

```
    # Sắp xếp kết quả theo thứ tự tăng dần (độ dài ngắn nhất đến dài nhất)
    # Tùy chọn: Nếu muốn chuỗi con, bạn có thể trích xuất S[0:length]
    results.sort()
    return results
# ---------------- CHẠY THỬ ----------------
P1 = "ababa"
# LPS[4] = 3 ("aba"). LPS[2] = 1 ("a").
print(f"P='{P1}' | Do dai tien to/hau to: {find_all_prefix_suffixes(P1)}")
# Kết quả: [1, 3]
P2 = "aabaaac"
# LPS[6] = 3 ("aac"). LPS[2] = 0.
print(f"P='{P2}' | Do dai tien to/hau to: {find_all_prefix_suffixes(P2)}")
# Kết quả: [1, 3] (Lưu ý: LPS[6] là 3 "aac", LPS[2] là 0. Có lỗi trong việc tính P2
ở câu trước. P2 = "aabaaac" -> LPS=[0, 1, 0, 1, 2, 2, 3])
P3 = "abcab"
# LPS[4] = 2 ("ab"). LPS[1] = 0.
print(f"P='{P3}' | Do dai tien to/hau to: {find_all_prefix_suffixes(P3)}")
# Kết quả: [2]
```

=> Kết quả sau khi chạy chương trình:

```
P='ababa' | Do dai tien to/hau to: [1, 3]
P='aabaaac' | Do dai tien to/hau to: []
P='abcab' | Do dai tien to/hau to: [2]
```

2.2. Code C++:

```
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
#include <string>            // Thư viện xử lí xâu
using namespace std;         // Tiền tố std
// Ham xay dung mang tien to dai nhat cung la hau to (LPS Array)
vector<int> compute_lps_array(const string &P)
{
    // Ham xay dung mang tien to dai nhat cung la hau to (LPS Array)
    int M = P.length();
    vector<int> lps(M, 0);
    int length = 0;
    int i = 1;
    while (i < M)
    {
        if (P[i] == P[length])
        {
            length++;
            lps[i] = length;
            i++;
        }
        else
        {
            if (length != 0)
            {
                length = lps[length - 1];
            }
            else
            {
                lps[i] = 0;
                i++;
            }
        }
    }
    return lps;
```

```
}
// Ham tim tat ca cac tien to vua la hau to
vector<int> find_all_prefix_suffixes(const string &P)
{
    // Su dung mang LPS de tim tat ca cac tien to cua P ma cung la hau to cua P.
    // Tra ve danh sach cac DO DAI cua cac tien to/hau to nay.
    int M = P.length();
    if (M == 0)
    {
        return {};
    }
    // Buoc 1: Tien xu ly de co duoc mang LPS
    vector<int> lps = compute_lps_array(P);
    vector<int> results;
    // Bat dau voi do dai tien to/hau to dai nhat cua toan bo xau P.
    // LPS[M-1] la do dai cua tien to dai nhat cua P[0..M-1] ma cung la hau to cua
no.
    int length = lps[M - 1];
    // Buoc 2: Truy vet nguoc (Backtracking)
    while (length > 0)
    {
        // Them do dai hien tai vao danh sach
        results.push_back(length);
        // Nhay ve do dai tien to/hau to dai nhat cua tien to/hau to hien tai
        // Vi du: Neu P = "ababa" (M=5), LPS[4]=3 ("aba").
        // Tiep theo, ta kiem tra LPS[3-1] = LPS[2] = 1 ("a").
        length = lps[length - 1];
    }
    // Sap xep ket qua theo thu tu tang dan (do dai ngan nhat den dai nhat)
    // Tuy chon: Neu muon chuoi con, ban co the trich xuat P.substr(0, length)
    sort(results.begin(), results.end());
    return results;
}
int main()
{
    // ----------------- CHAY THU -----------------
    string P1 = "ababa";
    // LPS[4] = 3 ("aba"). LPS[2] = 1 ("a").
    vector<int> res1 = find_all_prefix_suffixes(P1);
    cout << "P='" << P1 << "' | Do dai tien to/hau to: ";
    for (int x : res1) cout << x << " ";
    cout << endl;
    // Ket qua: [1, 3]
    string P2 = "aabaaac";
    // LPS[6] = 3 ("aac"). LPS[2] = 0.
    vector<int> res2 = find_all_prefix_suffixes(P2);
    cout << "P='" << P2 << "' | Do dai tien to/hau to: ";
    for (int x : res2) cout << x << " ";
    cout << endl;
    // Ket qua: [1, 3]
    // (Luu y: P2 = "aabaaac" -> LPS = [0, 1, 0, 1, 2, 2, 3])
    string P3 = "abcab";
    // LPS[4] = 2 ("ab"). LPS[1] = 0.
    vector<int> res3 = find_all_prefix_suffixes(P3);
    cout << "P='" << P3 << "' | Do dai tien to/hau to: ";
    for (int x : res3) cout << x << " ";
    cout << endl;
    // Ket qua: [2]
```

```
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
P='ababa' | Do dai tien to/hau to: 1 3
P='aabaaac' | Do dai tien to/hau to:
P='abcab' | Do dai tien to/hau to: 2
```

BÀI 18. MÃ HÓA VÀ XỬ LÝ KÝ TỰ SỐ

I. Mục Tiêu:

- **Kiến thức:** HS nắm vững cách chuyển đổi qua lại giữa ký tự ('0'..'9', 'a'..'z') và giá trị số (0..9, 1..26) trong cả Python và C++. Hiểu rõ cơ chế của Mã hóa Caesar và phép cộng/trừ số lớn bằng xâu.
- **Kỹ năng:** HS cài đặt được các hàm chuyển đổi ký tự-số. Áp dụng phép toán modulo cho mã hóa. Thực hiện phép cộng số lớn bằng xâu.
- **Tư duy:** Rèn luyện khả năng xử lý biên (ví dụ: tràn số, ký tự không hợp lệ) và áp dụng vòng lặp để thực hiện các phép toán số học phức tạp.

II. Phân Bổ Thời Gian (90 Phút):

Hoạt Động	Nội Dung	Thời gian (Phút)
Ôn tập	Ôn tập KMP, Độ phức tạp	10
Lý thuyết 1	Chuyển đổi Ký tự sang Số và ngược lại	10
Thực hành 1	Bài 1. Mã hóa Caesar (Cipher)	30
Lý thuyết 2	Bài toán Số học Xâu (String Arithmetic)	10
Thực hành 2	Bài 2. Cộng hai Số lớn bằng Xâu	25
Tổng kết	Bài tập về nhà, Củng cố	5
Tổng cộng		**90**

III. Lý Thuyết 1: Chuyển đổi Ký tự - Số (10 phút)

1. Ký tự Số ('0' đến '9'):

- **Mục đích:** Chuyển đổi ký tự đại diện cho chữ số thành giá trị số nguyên tương ứng (0-9).
- **Kỹ thuật:** Dựa trên Bảng mã ASCII/Unicode, các ký tự số được sắp xếp liên tiếp.
  + **Ký tự sang Số:** Lấy mã ASCII của ký tự và trừ đi mã ASCII của ký tự '0':
    value = ord('digit') - ord('0')
  + **Số sang Ký tự:** Lấy giá trị số và cộng với mã ASCII của ký tự '0':
    char = chr(value + ord('0'))

2. Ký tự Chữ cái ('a' đến 'z'):

- **Mục đích:** Dùng trong các bài toán mã hóa/giải mã như Caesar Cipher.
- **Kỹ thuật:** Tương tự như trên, nhưng sử dụng 'a' (hoặc 'A') làm cơ sở.
  + **Ký tự sang Giá trị (0-25):** value = ord('char') - ord('a')
  + **Giá trị (0-25) sang Ký tự:** char = chr(value + ord('a'))

IV. Thực Hành 1: Mã hóa Caesar (30 phút)

1. Bài 1. Mã hóa Caesar cơ bản (chỉ xử lý chữ cái thường)

1.1. Code Python:

```python
def caesar_cipher(text, shift):
    result = []
    # Chuẩn hóa shift để tránh số quá lớn hoặc quá nhỏ
    shift = shift % 26
    for char in text:
        if 'a' <= char <= 'z':
            # 1. Chuyển ký tự thành giá trị 0-25
            original_value = ord(char) - ord('a')
            # 2. Áp dụng phép dịch chuyển (dùng phép cộng và modulo)
            # new_value = (original_value + shift) mod 26
            # (+ 26) đảm bảo kết quả dương khi dùng phép dịch chuyển âm.
            new_value = (original_value + shift + 26) % 26
            # 3. Chuyển giá trị 0-25 thành ký tự mới
            new_char = chr(new_value + ord('a'))
            result.append(new_char)
        else:
            # Giữ nguyên các ký tự khác (khoảng trắng, dấu, số)
            result.append(char)
    # Lệnh return PHẢI NẰM NGOÀI vòng lặp for
    return "".join(result)
# Chạy thử
S = "hello world"
SHIFT = 3 # Dịch chuyển 3
# Kết quả: "khoor zruog"
print(f"Ma hoa: {caesar_cipher(S, SHIFT)}")
```

=> Kết quả sau khi chạy chương trình:

```
Ma hoa: khoor zruog
```

- Giải thích Code Python:

Phép Modulo (mod 26): Đây là chìa khóa để đảm bảo việc dịch chuyển lặp lại từ 'z' về 'a'. Việc thêm + 26 là cần thiết nếu **shift** là số âm (ví dụ: giải mã).

1.2. Code C++:

```cpp
#include <iostream>         // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>    // Thư viện tổng hợp
#include <string>           // Thư viện xử lí xâu
using namespace std;        // Tiền tố std
string caesar_cipher(const string &text, int shift)
{
    string result = "";
    // Chuan hoa shift de tranh so qua lon hoac qua nho
    shift = shift % 26;
    for (char ch : text)
    {
        if (ch >= 'a' && ch <= 'z')
        {
            // 1. Chuyen ky tu thanh gia tri 0-25
            int original_value = ch - 'a';
            // 2. Ap dung phep dich chuyen (dung phep cong va modulo)
            // new_value = (original_value + shift) mod 26
            // (+ 26) dam bao ket qua duong khi dung phep dich chuyen am.
            int new_value = (original_value + shift + 26) % 26;
            // 3. Chuyen gia tri 0-25 thanh ky tu moi
            char new_char = char(new_value + 'a');
```

```
            result += new_char;
        }
        else
        {
            // Giu nguyen cac ky tu khac (khoang trang, dau, so)
            result += ch;
        }
    }
    // Lenh return PHAI NAM NGOAI vong lap for
    return result;
}
int main()
{
    // Chay thu
    string S = "hello world";
    int SHIFT = 3; // Dich chuyen 3
    // Ket qua: "khoor zruog"
    cout << "Ma hoa: " << caesar_cipher(S, SHIFT) << endl;
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
Ma hoa: khoor zruog
```

- Giải thích Code C++:

Casting (Ép kiểu): C++ tự động ép kiểu khi thực hiện c - 'a', nhưng phải cẩn thận khi cộng new_value + 'a' để đảm bảo kết quả là char. Cần dùng (shift % 26 + 26) % 26 để xử lý shift âm một cách an toàn.

V. Lý thuyết 2: Bài toán Số học Xâu (String Arithmetic) (10 phút)

1. Tại sao cần Số học Xâu?

Các ngôn ngữ lập trình (Python, C++) có giới hạn về kiểu dữ liệu số nguyên (int, long long). Nếu cần thực hiện phép tính với các số có hơn 18-19 chữ số (ví dụ: 100 chữ số), ta phải lưu trữ chúng dưới dạng **xâu (string)** và tự cài đặt các thuật toán số học cơ bản (cộng, trừ, nhân).

2. Kỹ thuật Cộng Số lớn (Big Integer Addition)

Kỹ thuật cộng số lớn bằng xâu mô phỏng lại cách chúng ta thực hiện phép cộng trên giấy:

- **Bước 1. Đảo ngược Xâu:** Đảo ngược cả hai xâu số để bắt đầu cộng từ chữ số hàng đơn vị (chỉ mục 0).
- **Bước 2. Duyệt và Cộng:** Duyệt qua các xâu từ trái sang phải, thực hiện phép cộng hai chữ số ở vị trí hiện tại và thêm **số nhớ** (carry).
- **Bước 3. Xử lý số nhớ (Carry):**
  + Tổng = digit1 + digit2 + carry
  + Chữ số kết quả = Tổng mod 10
  + Số nhớ mới = Tổng div 10
- **Bước 4. Hoàn thành:** Sau khi hết chữ số, nếu vẫn còn số nhớ cuối cùng (carry > 0), phải thêm nó vào kết quả.
- **Bước 5. Đảo ngược Kết quả:** Đảo ngược xâu kết quả trở lại.

VI. Thực Hành 2: Cộng hai Số lớn bằng Xâu (25 phút)

1. Bài 2. Cộng hai Số lớn được đại diện bởi Xâu

1.1. Code Python:

```python
def add_big_numbers(num1, num2):
    # Đảo ngược xâu
    n1 = num1[::-1]
    n2 = num2[::-1]
    result = []
    i, j = 0, 0
    carry = 0
    # Vòng lặp: Tiếp tục miễn là còn số nhớ hoặc còn chữ số trong n1/n2
    while i < len(n1) or j < len(n2) or carry:
        # Lấy giá trị số từ ký tự (sửa lỗi cú pháp gán biến)
        # 1. Lấy d1 (Nếu i còn trong phạm vi, lấy giá trị; ngược lại, là 0)
        if i < len(n1):
            d1 = int(n1[i])
        else:
            d1 = 0
        # 2. Lấy d2 (Nếu j còn trong phạm vi, lấy giá trị; ngược lại, là 0)
        if j < len(n2):
            d2 = int(n2[j])
        else:
            d2 = 0
        # Phép cộng (Phần này phải nằm TRONG vòng lặp while)
        total = d1 + d2 + carry
        # Cập nhật số nhớ và chữ số kết quả
        carry = total // 10     # Số nhớ mới
        digit = total % 10      # Chữ số ở hàng hiện tại
        # Thêm chữ số (chuyển lại thành ký tự)
        result.append(str(digit))
        # Tăng chỉ mục (Phần này phải nằm TRONG vòng lặp while)
        i += 1
        j += 1
    # Đảo ngược kết quả cuối cùng (Phần này phải nằm TRONG hàm)
    return "".join(result)[::-1]
# Chạy thử
A = "12345678901234567890"  # 20 chữ số
B = "98765432109876543210"  # 20 chữ số
# Kết quả mong đợi: 111111111011111111100 (21 chữ số)
print(f"Tong: {add_big_numbers(A, B)}")
```

=> Kết quả sau khi chạy chương trình:

```
Tong: 111111111011111111100
```

- Giải thích Code Python:

+ Kỹ thuật Đảo ngược: Python dùng **[::-1]** để đảo ngược, giúp việc xử lý chỉ mục dễ dàng hơn (chỉ mục 0 luôn là hàng đơn vị).

+ **Ép kiểu:** Cần dùng **int()** và **str()** để chuyển đổi giữa ký tự và số trong vòng lặp.

1.2. Code C++:

```cpp
#include <iostream>         // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>    // Thư viện tổng hợp
#include <string>           // Thư viện xử lí xâu
using namespace std;        // Tiền tố std
string add_big_numbers(const string &num1, const string &num2)
{
    // Dao nguoc xau
    string n1 = string(num1.rbegin(), num1.rend());
    string n2 = string(num2.rbegin(), num2.rend());
    string result = "";
```

```
    int i = 0, j = 0;
    int carry = 0;
    // Vong lap: Tiep tuc mien la con so nho hoac con chu so trong n1/n2
    while (i < (int)n1.length() || j < (int)n2.length() || carry)
    {
        // Lay gia tri so tu ky tu (sua loi cu phap gan bien)
        // 1. Lay d1 (Neu i con trong pham vi, lay gia tri; nguoc lai, la 0)
        int d1;
        if (i < (int)n1.length())
        {
            d1 = n1[i] - '0';
        }
        else
        {
            d1 = 0;
        }
        // 2. Lay d2 (Neu j con trong pham vi, lay gia tri; nguoc lai, la 0)
        int d2;
        if (j < (int)n2.length())
        {
            d2 = n2[j] - '0';
        }
        else
        {
            d2 = 0;
        }
        // Phep cong (Phan nay phai nam TRONG vong lap while)
        int total = d1 + d2 + carry;
        // Cap nhat so nho va chu so ket qua
        carry = total / 10;     // So nho moi
        int digit = total % 10; // Chu so o hang hien tai
        // Them chu so (chuyen lai thanh ky tu)
        result.push_back(char(digit + '0'));
        // Tang chi muc (Phan nay phai nam TRONG vong lap while)
        i++;
        j++;
    }
    // Dao nguoc ket qua cuoi cung (Phan nay phai nam TRONG ham)
    reverse(result.begin(), result.end());
    return result;
}
int main()
{
    // Chay thu
    string A = "12345678901234567890";  // 20 chu so
    string B = "98765432109876543210";  // 20 chu so
    // Ket qua mong doi: 111111111011111111100 (21 chu so)
    cout << "Tong: " << add_big_numbers(A, B) << endl;
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
Tong: 111111111011111111100
```

- Giải thích Code C++:

Kỹ thuật Đảo ngược: C++ dùng **string n1 = string(num1.rbegin(), num1.rend());** và **reverse(result.begin(), result.end());** để đảo ngược, giúp việc xử lý chỉ mục dễ dàng hơn.

VII. Bài Tập Về Nhà & Tổng Kết (5 phút)

1. Bài tập 1 (Phép Trừ Số lớn): Cài đặt hàm trừ hai số lớn (giả định số bị trừ luôn lớn hơn hoặc bằng số trừ).

1.1. Code Python:

```python
def subtract_big_numbers(num_a, num_b):
    # Trừ hai số lớn (num_a - num_b), giả định num_a >= num_b.
    # Độ phức tạp: O(N), với N là độ dài của num_a.
    # Đảo ngược xâu: Bắt đầu từ hàng đơn vị
    a_rev = num_a[::-1]
    b_rev = num_b[::-1]
    result = []
    i = 0
    borrow = 0  # Số mượn
    # Vòng lặp: Duyệt qua các chữ số của số bị trừ (A)
    while i < len(a_rev):
        # 1. Lấy chữ số A: digit_A
        digit_a = int(a_rev[i])
        # 2. Lấy chữ số B: digit_B
        # Nếu đã hết chữ số trong B, coi như 0
        if i < len(b_rev):
            digit_b = int(b_rev[i])
        else:
            digit_b = 0
        # 3. Phép trừ cơ bản: difference = digit_A - digit_B - borrow
        difference = digit_a - digit_b - borrow
        # 4. Xử lý Mượn (Borrow)
        if difference < 0:
            # Nếu kết quả âm, ta phải mượn 1 từ hàng tiếp theo
            # Đảm bảo chữ số kết quả nằm trong khoảng 0-9
            digit_result = 10 + difference
            borrow = 1 # Thiết lập mượn cho vòng lặp tiếp theo
        else:
            digit_result = difference
            borrow = 0 # Không cần mượn
        # Thêm chữ số kết quả (chuyển thành ký tự)
        result.append(str(digit_result))
        i += 1
    # 5. Đảo ngược kết quả cuối cùng
    final_result = "".join(result)[::-1]
    # 6. Loại bỏ số 0 không cần thiết ở đầu (ví dụ: '00123' -> '123')
    # Trừ khi kết quả là 0
    if final_result == "":
        return "0"
    return final_result.lstrip('0') or '0'
# ----------------- CHẠY THỬ -----------------
# Ví dụ 1: Không có mượn
A1 = "98765"
B1 = "12345"
# Kết quả: 86420
print(f"A1 - B1 = {subtract_big_numbers(A1, B1)}")
# Ví dụ 2: Có mượn
A2 = "1000"
B2 = "1"
# Kết quả: 0999 -> 999
print(f"A2 - B2 = {subtract_big_numbers(A2, B2)}")
# Ví dụ 3: Cùng độ dài và cần mượn nhiều
```

```
A3 = "5000"
B3 = "4999"
# Kết quả: 0001 -> 1
print(f"A3 - B3 = {subtract_big_numbers(A3, B3)}")
# Ví dụ 4: Trừ hai số bằng nhau
A4 = "123456789"
B4 = "123456789"
# Kết quả: 0
print(f"A4 - B4 = {subtract_big_numbers(A4, B4)}")
```

=> Kết quả sau khi chạy chương trình:

```
A1 - B1 = 86420
A2 - B2 = 999
A3 - B3 = 1
A4 - B4 = 0
```

1.2. Code C++:

```
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
#include <string>            // Thư viện xử lí xâu
using namespace std;         // Tiền tố std
string subtract_big_numbers(const string &num_a, const string &num_b)
{
    // Tru hai so lon (num_a - num_b), gia dinh num_a >= num_b.
    // Do phuc tap: O(N), voi N la do dai cua num_a.
    // Dao nguoc xau: Bat dau tu hang don vi
    string a_rev(num_a.rbegin(), num_a.rend());
    string b_rev(num_b.rbegin(), num_b.rend());
    string result = "";
    int i = 0;
    int borrow = 0;  // So muon
    // Vong lap: Duyet qua cac chu so cua so bi tru (A)
    while (i < (int)a_rev.length())
    {
        // 1. Lay chu so A: digit_a
        int digit_a = a_rev[i] - '0';
        // 2. Lay chu so B: digit_b
        // Neu da het chu so trong B, coi nhu 0
        int digit_b;
        if (i < (int)b_rev.length())
        {
            digit_b = b_rev[i] - '0';
        }
        else
        {
            digit_b = 0;
        }
        // 3. Phep tru co ban: difference = digit_A - digit_B - borrow
        int difference = digit_a - digit_b - borrow;
        int digit_result;
        // 4. Xu ly Muon (Borrow)
        if (difference < 0)
        {
            // Neu ket qua am, ta phai muon 1 tu hang tiep theo
            // Dam bao chu so ket qua nam trong khoang 0-9
            digit_result = 10 + difference;
            borrow = 1; // Thiet lap muon cho vong lap tiep theo
        }
        else
```

```
        {
            digit_result = difference;
            borrow = 0; // Khong can muon
        }
        // Them chu so ket qua (chuyen thanh ky tu)
        result.push_back(char(digit_result + '0'));
        i++;
    }
    // 5. Dao nguoc ket qua cuoi cung
    reverse(result.begin(), result.end());
    string final_result = result;
    // 6. Loai bo so 0 khong can thiet o dau (vi du: "00123" -> "123")
    // Tru khi ket qua la 0
    if (final_result == "")
    {
        return "0";
    }
    int pos = 0;
    while (pos < (int)final_result.length() && final_result[pos] == '0')
    {
        pos++;
    }
    if (pos == (int)final_result.length())
    {
        return "0";
    }
    return final_result.substr(pos);
}
int main()
{
    // ----------------- CHAY THU -----------------
    // Vi du 1: Khong co muon
    string A1 = "98765";
    string B1 = "12345";
    // Ket qua: 86420
    cout << "A1 - B1 = " << subtract_big_numbers(A1, B1) << endl;
    // Vi du 2: Co muon
    string A2 = "1000";
    string B2 = "1";
    // Ket qua: 0999 -> 999
    cout << "A2 - B2 = " << subtract_big_numbers(A2, B2) << endl;
    // Vi du 3: Cung do dai va can muon nhieu
    string A3 = "5000";
    string B3 = "4999";
    // Ket qua: 0001 -> 1
    cout << "A3 - B3 = " << subtract_big_numbers(A3, B3) << endl;
    // Vi du 4: Tru hai so bang nhau
    string A4 = "123456789";
    string B4 = "123456789";
    // Ket qua: 0
    cout << "A4 - B4 = " << subtract_big_numbers(A4, B4) << endl;
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
A1 - B1 = 86420
A2 - B2 = 999
A3 - B3 = 1
A4 - B4 = 0
```

2. Bài tập 2 (Mã hóa Vigenère): Mở rộng Mã hóa Caesar thành Mã hóa Vigenère, sử dụng một xâu khóa (key) để thay đổi độ dịch chuyển liên tục.

2.1. Code Python:

```
def vigenere_cipher(text, key):
    # Mã hóa văn bản bằng thuật toán Vigenère.
    # Chỉ xử lý chữ cái thường (a-z).
    # Độ phức tạp: O(N), với N là độ dài của văn bản gốc.
    # 1. Chuẩn hóa và chuyển khóa thành chuỗi các giá trị dịch chuyển (0-25)
    # Loại bỏ ký tự không phải chữ cái và chuyển về số
    key_values = []
    for char in key.lower(): # Đảm bảo khóa là chữ thường để tính toán
        if 'a' <= char <= 'z':
            key_values.append(ord(char) - ord('a'))
    if not key_values:
        # Xử lý trường hợp khóa rỗng hoặc không hợp lệ
        return text
    result = []
    key_index = 0
    key_length = len(key_values)
    for char in text:
        # Chỉ xử lý chữ cái thường
        if 'a' <= char <= 'z':
            # 2. Lấy độ dịch chuyển hiện tại (Shift)
            # Lấy giá trị khóa tại vị trí key_index (sử dụng modulo để lặp lại
khóa)
            shift = key_values[key_index % key_length]
            # 3. Mã hóa Caesar
            original_value = ord(char) - ord('a')
            # (original_value + shift) mod 26
            new_value = (original_value + shift) % 26
            new_char = chr(new_value + ord('a'))
            result.append(new_char)
            # CHÌA KHÓA: Chỉ tăng chỉ mục khóa khi có sự mã hóa
            key_index += 1
        else:
            # Giữ nguyên các ký tự không phải chữ cái
            result.append(char)
    return "".join(result)
# ----------------- CHẠY THỬ -----------------
TEXT = "attack at dawn"
KEY = "lemon"
# Quy ước:
# T: a t t a c k   a t   d a w n
# K: l e m o n l e m o n l e m o n
# S: 11 4 12 14 13 11 4 12 14 13 11 4 12 14 13 (độ dịch chuyển)
# C: l x w a a p   l x s s a y
print(f"Van ban goc: {TEXT}")
print(f"Khoa: {KEY}")
# Kết quả mong đợi: lxwaap lxssay
print(f"Ma hoa Vigenere: {vigenere_cipher(TEXT, KEY)}")
# Ví dụ 2: Khóa ngắn, cần lặp lại
T2 = "programming"
```

```
K2 = "can"
# Kết quả: rrbirnomvpg
print(f"Ma hoa Vigenere: {vigenere_cipher(T2, K2)}")
```

=> Kết quả sau khi chạy chương trình:

```
Van ban goc: attack at dawn
Khoa: lemon
Ma hoa Vigenere: lxfopv ef rnhr
Ma hoa Vigenere: rrbirnomvpg
```

2.2. Code C++:

```
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
#include <string>            // Thư viện xử lí xâu
using namespace std;         // Tiền tố std
string vigenere_cipher(const string &text, const string &key)
{
    // Ma hoa van ban bang thuat toan Vigenere.
    // Chi xu ly chu cai thuong (a-z).
    // Do phuc tap: O(N), voi N la do dai cua van ban goc.
    // 1. Chuan hoa va chuyen khoa thanh chuoi cac gia tri dich chuyen (0-25)
    // Loai bo ky tu khong phai chu cai va chuyen ve so
    vector<int> key_values;
    for (char ch : key)
    {
        char c = tolower(ch); // Dam bao khoa la chu thuong de tinh toan
        if (c >= 'a' && c <= 'z')
        {
            key_values.push_back(c - 'a');
        }
    }
    if (key_values.empty())
    {
        // Xu ly truong hop khoa rong hoac khong hop le
        return text;
    }
    string result = "";
    int key_index = 0;
    int key_length = key_values.size();
    for (char ch : text)
    {
        // Chi xu ly chu cai thuong
        if (ch >= 'a' && ch <= 'z')
        {
            // 2. Lay do dich chuyen hien tai (Shift)
            // Lay gia tri khoa tai vi tri key_index (su dung modulo de lap lai
khoa)
            int shift = key_values[key_index % key_length];
            // 3. Ma hoa Caesar
            int original_value = ch - 'a';
            // (original_value + shift) mod 26
            int new_value = (original_value + shift) % 26;
            char new_char = char(new_value + 'a');
            result.push_back(new_char);
            // CHIA KHOA: Chi tang chi muc khoa khi co su ma hoa
            key_index++;
        }
        else
        {
```

```
            // Giu nguyen cac ky tu khong phai chu cai
            result.push_back(ch);
        }
    }
    return result;
}
int main()
{
    // ----------------- CHAY THU -----------------
    string TEXT = "attack at dawn";
    string KEY = "lemon";
    // Quy uoc:
    // T: a t t a c k   a t   d a w n
    // K: l e m o n l e m o n l e m o n
    // S: 11 4 12 14 13 11 4 12 14 13 11 4 12 14 13 (do dich chuyen)
    // C: l x w a a p   l x s s a y
    cout << "Van ban goc: " << TEXT << endl;
    cout << "Khoa: " << KEY << endl;
    // Ket qua mong doi: lxwaap lxssay
    cout << "Ma hoa Vigenere: " << vigenere_cipher(TEXT, KEY) << endl;
    // Vi du 2: Khoa ngan, can lap lai
    string T2 = "programming";
    string K2 = "can";
    // Ket qua: rrbirnomvpg
    cout << "Ma hoa Vigenere: " << vigenere_cipher(T2, K2) << endl;
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
Van ban goc: attack at dawn
Khoa: lemon
Ma hoa Vigenere: lxfopv ef rnhr
Ma hoa Vigenere: rrbirnomvpg
```

CHỦ ĐỀ 7. SẮP XẾP & TÌM KIẾM

BÀI 19. CÁC THUẬT TOÁN SẮP XẾP CƠ BẢN: NỔI BỌT, CHỌN, CHÈN

I. Mục Tiêu:

- **Kiến thức:** HS hiểu rõ nguyên lý hoạt động, cơ chế hoán đổi và độ phức tạp của ba thuật toán sắp xếp cơ bản: Bubble Sort, Selection Sort, và Insertion Sort.
- **Kỹ năng:** HS cài đặt được cả ba thuật toán bằng Python và C++. HS phân tích được hiệu suất của chúng trong các trường hợp tốt nhất, xấu nhất và trung bình.
- **Tư duy:** HS nhận biết được sự khác biệt giữa các thuật toán dựa trên số lần **hoán đổi** (swaps) và số lần **so sánh** (comparisons).

II. Phân Bổ Thời Gian (90 Phút):

Hoạt Động	Nội Dung	Thời gian (Phút)
Ôn tập	Ôn tập KMP, Mã hóa Vigenere	10
Lý thuyết 1	Bubble Sort: Cơ chế và Độ phức tạp	15

Thực hành 1	Bài 1. Cài đặt Bubble Sort	15
Lý thuyết 2	Selection Sort & Insertion Sort: Cơ chế	15
Thực hành 2	Bài 2. Cài đặt Selection Sort	15
Thực hành 3	Bài 3. Cài đặt Insertion Sort	15
Tổng kết	So sánh 3 thuật toán, Bài tập về nhà	5
Tổng cộng		**90**

III. Lý Thuyết 1: Bubble Sort (15 phút)

1. Nguyên lý Hoạt động:

Bubble Sort (Sắp xếp Nổi bọt) là thuật toán đơn giản nhất:

- Bước 1. Lặp qua danh sách nhiều lần.
- Bước 2. Trong mỗi lần lặp, so sánh các cặp phần tử kề nhau.
- Bước 3. Nếu chúng không đúng thứ tự (phần tử bên trái lớn hơn phần tử bên phải), thực hiện **hoán đổi** (swap).
- Bước 4. Mỗi lần lặp sẽ "đẩy" phần tử lớn nhất còn lại về vị trí cuối cùng đúng của nó (như bong bóng nổi lên).

2. Độ phức tạp Thời gian

- **Trung bình/Xấu nhất:** $O(N^2)$. Xấu nhất khi mảng đảo ngược (cần nhiều hoán đổi nhất).
- **Tốt nhất:** O(N) (Khi mảng đã được sắp xếp, có thể tối ưu bằng cách kiểm tra xem có hoán đổi nào xảy ra trong lần lặp đó không).
- **Không gian:** O(1) (In-place).

IV. Thực Hành 1: Cài đặt Bubble Sort (15 phút)

1. Code Python:

```python
def bubble_sort(arr):
    n = len(arr)
    # Lặp qua tất cả các phần tử (n lần tối đa)
    for i in range(n):
        swapped = False  # Đặt lại cờ hoán đổi cho mỗi lần lặp ngoài
        # Vòng lặp trong: So sánh và đẩy phần tử lớn nhất về cuối
        # j chạy đến n - 1 - i vì i phần tử cuối cùng đã được sắp xếp
        for j in range(0, n - i - 1):
            if arr[j] > arr[j + 1]:
                # Hoán đổi
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
                swapped = True
        # KIỂM TRA TỐI ƯU HÓA: Phải nằm SAU vòng lặp for j
        # và NẰM TRONG vòng lặp for i
        if not swapped:
            # Nếu không có hoán đổi nào trong lần lặp này, mảng đã sắp xếp
            break
    return arr
# Chạy thử
arr = [64, 34, 25, 12, 22, 11, 90]
print(bubble_sort(arr))
# Kết quả: [11, 12, 22, 25, 34, 64, 90]
```

=> Kết quả sau khi chạy chương trình:

```
[11, 12, 22, 25, 34, 64, 90]
```

- Giải thích Code Python:

Tối ưu hóa: Sử dụng biến **swapped**. Nếu không có hoán đổi nào trong một lượt, mảng đã sắp xếp và ta có thể thoát khỏi vòng lặp **for i** (O(N) trong trường hợp tốt nhất).

2. Code C++:

```
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
using namespace std;         // Tiền tố std
vector<int> bubble_sort(vector<int> arr)
{
    int n = arr.size();
    // Lap qua tat ca cac phan tu (n lan toi da)
    for (int i = 0; i < n; i++)
    {
        bool swapped = false;  // Dat lai co hoan doi cho moi lan lap ngoai
        // Vong lap trong: So sanh va day phan tu lon nhat ve cuoi
        // j chay den n - 1 - i vi i phan tu cuoi cung da duoc sap xep
        for (int j = 0; j < n - i - 1; j++)
        {
            if (arr[j] > arr[j + 1])
            {
                // Hoan doi
                swap(arr[j], arr[j + 1]);
                swapped = true;
            }
        }
        // KIEM TRA TOI UU HOA: Phai nam SAU vong lap for j
        // va NAM TRONG vong lap for i
        if (!swapped)
        {
            // Neu khong co hoan doi nao trong lan lap nay, mang da sap xep
            break;
        }
    }
    return arr;
}
int main()
{
    // Chay thu
    vector<int> arr = {64, 34, 25, 12, 22, 11, 90};
    vector<int> sorted_arr = bubble_sort(arr);
    for (int x : sorted_arr)
    {
        cout << x << " ";
    }
    cout << endl;
    // Ket qua: 11 12 22 25 34 64 90
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
11 12 22 25 34 64 90
```

- Giải thích Code C++:

Hoán đổi: C++ sử dụng hàm **swap** có sẵn. Logic hai vòng lặp lồng nhau và tối ưu hóa **swapped** tương tự như Python.

V. Lý thuyết 2: Selection Sort & Insertion Sort (15 phút)

1. Selection Sort (Sắp xếp Chọn):

- **Ý tưởng:**

+ Chia mảng thành hai phần: phần đã sắp xếp (bên trái, ban đầu rỗng) và phần chưa sắp xếp (bên phải).

+ Lặp qua phần chưa sắp xếp, **tìm phần tử nhỏ nhất**.

+ Thực hiện **hoán đổi duy nhất** để đưa phần tử nhỏ nhất đó về vị trí đầu tiên của phần chưa sắp xếp (tức là vị trí cuối cùng của phần đã sắp xếp).

+ Lặp lại cho đến khi mảng được sắp xếp.

- **Độ phức tạp:** $O(N^2)$ (Luôn luôn). Selection Sort có ít **hoán đổi** nhất (O(N) lần) nhưng số lần **so sánh** vẫn là $O(N^2)$.

- **Không gian:** O(1) (In-place).

2. Insertion Sort (Sắp xếp Chèn):

- **Ý tưởng:** Tương tự như sắp xếp một bộ bài trên tay.

+ Chia mảng thành hai phần: phần đã sắp xếp (bên trái, ban đầu là phần tử đầu tiên) và phần chưa sắp xếp (bên phải).

+ Lặp qua các phần tử trong phần chưa sắp xếp (gọi là key).

+ Lấy key và **chèn** nó vào vị trí đúng trong phần đã sắp xếp bằng cách dịch chuyển (shift) các phần tử lớn hơn nó sang phải.

- **Độ phức tạp:**

+ **Trung bình/Xấu nhất:** $O(N^2)$. Xấu nhất khi mảng đảo ngược (cần nhiều dịch chuyển nhất).

+ **Tốt nhất:** O(N) (Khi mảng đã được sắp xếp). Insertion Sort là thuật toán nhanh nhất cho các mảng **gần như đã sắp xếp**.

- **Không gian:** O(1) (In-place).

VI. Thực Hành 2 & 3: Cài đặt Selection và Insertion Sort (30 phút)

1. Bài 2. Cài đặt Selection Sort

1.1. Code Python:

```
def selection_sort(arr):
    n = len(arr)
    # Vòng lặp ngoài: Duyệt qua từng vị trí cần sắp xếp (từ 0 đến n-2)
    for i in range(n - 1): # Chỉ cần đến n-2 vì n-1 sẽ tự động đúng
        # Giả sử phần tử nhỏ nhất là phần tử đầu tiên của phần chưa sắp xếp
        min_idx = i
        # Vòng lặp trong: Tìm phần tử nhỏ nhất trong phần chưa sắp xếp (từ i+1 đến
n-1)
        for j in range(i + 1, n):
            if arr[j] < arr[min_idx]:
                min_idx = j # Cập nhật chỉ mục của phần tử nhỏ nhất
        # --- BƯỚC HOÁN ĐỔI ĐÚNG ---
        # Chỉ Hoán đổi DUY NHẤT một lần, sau khi đã tìm thấy min_idx chính xác.
        arr[i], arr[min_idx] = arr[min_idx], arr[i]
    return arr
# Chạy thử
arr = [64, 25, 12, 22, 11]
print(f"Mang goc: {arr}")
print(f"Sap xep Selection Sort: {selection_sort(arr)}")
# Kết quả đúng: [11, 12, 22, 25, 64]
```

=> Kết quả sau khi chạy chương trình:

```
Mang goc: [64, 25, 12, 22, 11]
Sap xep Selection Sort: [11, 12, 22, 25, 64]
```

- Giải thích Code Python:

Hoán đổi đơn: Selection Sort chỉ hoán đổi 1 lần sau khi vòng lặp trong kết thúc. Nó ưu tiên giảm số lần hoán đổi, không phải số lần so sánh.

1.2. Code C++:

```
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
using namespace std;         // Tiền tố std
vector<int> selection_sort(vector<int> arr)
{
    int n = arr.size();
    // Vong lap ngoai: Duyet qua tung vi tri can sap xep (tu 0 den n-2)
    for (int i = 0; i < n - 1; i++)
    {
        // Chi can den n-2 vi n-1 se tu dong dung
        // Gia su phan tu nho nhat la phan tu dau tien cua phan chua sap xep
        int min_idx = i;
        // Vong lap trong: Tim phan tu nho nhat trong phan chua sap xep (tu i+1 den
n-1)
        for (int j = i + 1; j < n; j++)
        {
            if (arr[j] < arr[min_idx])
            {
                min_idx = j; // Cap nhat chi muc cua phan tu nho nhat
            }
        }
        // --- BUOC HOAN DOI DUNG ---
        // Chi Hoan doi DUY NHAT mot lan, sau khi da tim thay min_idx chinh xac.
        swap(arr[i], arr[min_idx]);
    }
    return arr;
}
int main()
{
    // Chay thu
    vector<int> arr = {64, 25, 12, 22, 11};
    cout << "Mang goc: ";
    for (int x : arr) cout << x << " ";
    cout << endl;
    vector<int> sorted_arr = selection_sort(arr);
    cout << "Sap xep Selection Sort: ";
    for (int x : sorted_arr) cout << x << " ";
    cout << endl;
    // Ket qua dung: 11 12 22 25 64
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
Mang goc: 64 25 12 22 11
Sap xep Selection Sort: 11 12 22 25 64
```

- Giải thích Code C++:

Vòng lặp ngoài: Chỉ cần chạy đến n-2 vì phần tử cuối cùng (n-1) sẽ tự động đúng vị trí.

2. Bài 3. Cài đặt Insertion Sort:

2.1. Code Python:

```python
def insertion_sort(arr):
    # Vòng lặp ngoài: Duyệt qua các phần tử chưa được sắp xếp (từ index 1)
    for i in range(1, len(arr)):
        # 1. Khởi tạo:
        key = arr[i] # Phần tử cần chèn (bài)
        j = i - 1    # Vị trí cuối cùng của phần đã sắp xếp (các bài trên
tay)
        # 2. Dịch chuyển (Shifting):
        # Lặp lại miễn là j hợp lệ (>= 0) và key nhỏ hơn phần tử đang xét (arr[j])
        while j >= 0 and key < arr[j]:
            arr[j + 1] = arr[j] # Dịch chuyển phần tử lớn hơn sang phải
            j -= 1
        # 3. Chèn:
        # Chèn key vào vị trí đúng (vị trí j+1 sau khi vòng lặp while kết thúc)
        arr[j + 1] = key
    return arr
# Chạy thử
arr_test = [12, 11, 13, 5, 6]
print(f"Mang goc: {arr_test}")
print(f"Sap xep Insertion Sort: {insertion_sort(arr_test)}")
# Kết quả đúng: [5, 6, 11, 12, 13]
```

=> Kết quả sau khi chạy chương trình:

```
Mang goc: [12, 11, 13, 5, 6]
Sap xep Insertion Sort: [5, 6, 11, 12, 13]
```

- Giải thích Code Python:

Vòng lặp while: Thực hiện việc dịch chuyển (shifting), không phải hoán đổi. Thuật toán này hiệu quả khi ít phải dịch chuyển (mảng gần sắp xếp).

2.2. Code C++:

```cpp
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
using namespace std;         // Tiền tố std
vector<int> insertion_sort(vector<int> arr)
{
    // Vong lap ngoai: Duyet qua cac phan tu chua duoc sap xep (tu index 1)
    for (int i = 1; i < (int)arr.size(); i++)
    {
        // 1. Khoi tao:
        int key = arr[i]; // Phan tu can chen (bai)
        int j = i - 1;    // Vi tri cuoi cung cua phan da sap xep (cac bai tren
tay)
        // 2. Dich chuyen (Shifting):
        // Lap lai mien la j hop le (>= 0) va key nho hon phan tu dang xet (arr[j])
        while (j >= 0 && key < arr[j])
        {
            arr[j + 1] = arr[j]; // Dich chuyen phan tu lon hon sang phai
            j--;
        }
        // 3. Chen:
        // Chen key vao vi tri dung (vi tri j+1 sau khi vong lap while ket thuc)
        arr[j + 1] = key;
    }
    return arr;
}
```

```
int main()
{
    // Chay thu
    vector<int> arr_test = {12, 11, 13, 5, 6};
    cout << "Mang goc: ";
    for (int x : arr_test) cout << x << " ";
    cout << endl;
    vector<int> sorted_arr = insertion_sort(arr_test);
    cout << "Sap xep Insertion Sort: ";
    for (int x : sorted_arr) cout << x << " ";
    cout << endl;
    // Ket qua dung: 5 6 11 12 13
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
Mang goc: 12 11 13 5 6
Sap xep Insertion Sort: 5 6 11 12 13
```

- Giải thích Code C++:

Biến key: Biến này lưu trữ giá trị của phần tử cần chèn (arr[i]) để nó không bị mất trong quá trình dịch chuyển.

VII. Bài Tập Về Nhà & Tổng Kết (5 phút)

1. Tổng kết. So sánh 3 Thuật toán ($O(N^2)$):

Thuật toán	Trung bình/Xấu nhất	Tốt nhất	Hoán đổi (Worst Case)	Ứng dụng
Bubble Sort	$O(N^2)$	$O(N)$	$O(N^2)$	Giáo dục, Mảng gần sắp xếp
Selection Sort	$O(N^2)$	$O(N^2)$	$O(N)$ (ít nhất)	Khi chi phí Hoán đổi rất cao
Insertion Sort	$O(N^2)$	$O(N)$	$O(N^2)$	Mảng nhỏ, Mảng gần sắp xếp

2. Bài tập về nhà:

1. Bài tập 1 (Tìm kiếm Tuyến tính): Cài đặt hàm tìm kiếm tuyến tính (Linear Search) và phân tích độ phức tạp O(N).

1.1. Code Python:

```
def linear_search(arr, target):
    # Thực hiện tìm kiếm tuyến tính trên mảng (hoặc danh sách) arr
    # để tìm kiếm giá trị target.
    # Tham số:
    #   arr (list): Danh sách các phần tử.
    #   target (any): Giá trị cần tìm kiếm.
    # Trả về:
    # int: Chỉ mục (index) của phần tử target đầu tiên tìm thấy.
    #   Trả về -1 nếu không tìm thấy.

    # Duyệt qua từng phần tử của mảng từ đầu đến cuối
```

```python
    for i in range(len(arr)):
        # So sánh phần tử hiện tại với giá trị mục tiêu
        if arr[i] == target:
            # Nếu tìm thấy, trả về chỉ mục của nó
            return i
    # Nếu vòng lặp kết thúc mà không tìm thấy, trả về -1
    return -1
# ----------------- CHẠY THỬ -----------------
data = [5, 12, 9, 3, 15, 7, 12]
target1 = 15
target2 = 12
target3 = 100
print(f"Du lieu: {data}")
print("-" * 30)
# Trường hợp 1: Tìm thấy (vị trí cuối)
index1 = linear_search(data, target1)
print(f"Tim kiem {target1}: Index = {index1}")
# Trường hợp 2: Tìm thấy (vị trí lặp lại, trả về vị trí đầu tiên)
index2 = linear_search(data, target2)
print(f"Tim kiem {target2}: Index = {index2}")
# Trường hợp 3: Không tìm thấy
index3 = linear_search(data, target3)
print(f"Tim kiem {target3}: Index = {index3}")
```

=> Kết quả sau khi chạy chương trình:

```
Du lieu: [5, 12, 9, 3, 15, 7, 12]
------------------------------
Tim kiem 15: Index = 4
Tim kiem 12: Index = 1
Tim kiem 100: Index = -1
```

1.2. Code C++:

```cpp
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
using namespace std;         // Tiền tố std
int linear_search(const vector<int> &arr, int target)
{
    // Thuc hien tim kiem tuyen tinh tren mang (hoac danh sach) arr
    // de tim kiem gia tri target.
    // Tham so:
    //   arr (vector): Danh sach cac phan tu.
    //   target (int): Gia tri can tim kiem.
    // Tra ve:
    // int: Chi muc (index) cua phan tu target dau tien tim thay.
    //   Tra ve -1 neu khong tim thay.

    // Duyet qua tung phan tu cua mang tu dau den cuoi
    for (int i = 0; i < (int)arr.size(); i++)
    {
        // So sanh phan tu hien tai voi gia tri muc tieu
        if (arr[i] == target)
        {
            // Neu tim thay, tra ve chi muc cua no
            return i;
        }
    }
    // Neu vong lap ket thuc ma khong tim thay, tra ve -1
    return -1;
}
```

```
int main()
{
    // ----------------- CHAY THU -----------------
    vector<int> data = {5, 12, 9, 3, 15, 7, 12};
    int target1 = 15;
    int target2 = 12;
    int target3 = 100;
    cout << "Du lieu: ";
    for (int x : data) cout << x << " ";
    cout << endl;
    cout << string(30, '-') << endl;
    // Truong hop 1: Tim thay (vi tri cuoi)
    int index1 = linear_search(data, target1);
    cout << "Tim kiem " << target1 << ": Index = " << index1 << endl;
    // Truong hop 2: Tim thay (vi tri lap lai, tra ve vi tri dau tien)
    int index2 = linear_search(data, target2);
    cout << "Tim kiem " << target2 << ": Index = " << index2 << endl;
    // Truong hop 3: Khong tim thay
    int index3 = linear_search(data, target3);
    cout << "Tim kiem " << target3 << ": Index = " << index3 << endl;
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
Du lieu: [5, 12, 9, 3, 15, 7, 12]
------------------------------
Tim kiem 15: Index = 4
Tim kiem 12: Index = 1
Tim kiem 100: Index = -1
```

2. Bài tập 2 (Phân tích Selection Sort): Sửa đổi code Selection Sort để đếm chính xác số lần hoán đổi và số lần so sánh xảy ra khi sắp xếp một mảng bất kỳ.

2.1. Code Python:

```
def selection_sort_analysis(arr):
    # Thực hiện Selection Sort và đếm chính xác số lần so sánh và hoán đổi.
    n = len(arr)
    comparison_count = 0
    swap_count = 0
    # Vòng lặp ngoài: Duyệt qua từng vị trí cần sắp xếp (từ 0 đến n-2)
    # i chạy từ 0 đến n-2 (tổng cộng n-1 lần)
    for i in range(n - 1):
        # Giả sử phần tử nhỏ nhất là phần tử đầu tiên của phần chưa sắp xếp
        min_idx = i
        # Vòng lặp trong: Tìm phần tử nhỏ nhất trong phần chưa sắp xếp
        # j chạy từ i+1 đến n-1
        for j in range(i + 1, n):
            # 1. Đếm lần so sánh (So sánh arr[j] < arr[min_idx])
            comparison_count += 1
            if arr[j] < arr[min_idx]:
                min_idx = j # Cập nhật chỉ mục của phần tử nhỏ nhất
        # 2. Hoán đổi: Chỉ thực hiện 1 lần sau khi vòng lặp j kết thúc
        # Chỉ hoán đổi nếu phần tử nhỏ nhất không phải là arr[i]
        if min_idx != i:
            arr[i], arr[min_idx] = arr[min_idx], arr[i]
            swap_count += 1 # Đếm lần hoán đổi thực tế
    # Trả về mảng đã sắp xếp và số liệu phân tích
    return arr, comparison_count, swap_count
# ----------------- CHẠY THỬ -----------------
```

```python
# Ví dụ 1: Mảng ngẫu nhiên
arr1 = [64, 25, 12, 22, 11] # N = 5
sorted_arr1, comp1, swap1 = selection_sort_analysis(arr1)
print("--- Vi du 1: Mang ngau nhien ---")
print(f"Mang da sap xep: {sorted_arr1}")
# Dự đoán so sánh: 5*(5-1)/2 = 10
print(f"Tong so lan So sanh: {comp1}")
print(f"Tong so lan Hoan doi: {swap1}") # 4 hoán đổi thực tế
print("\n--- Vi du 2: Mang da sap xep (Best Case) ---")
# Ví dụ 2: Mảng đã sắp xếp (So sánh tối đa, hoán đổi tối thiểu)
arr2 = [10, 20, 30, 40, 50] # N = 5
sorted_arr2, comp2, swap2 = selection_sort_analysis(arr2)
print(f"Mang da sap xep: {sorted_arr2}")
print(f"Tong so lan So sanh: {comp2}") # Vẫn là 10 (luôn là O(N^2))
print(f"Tong so lan Hoan doi: {swap2}") # 0 hoán đổi thực tế (min_idx luôn bằng i)
print("\n--- Vi du 3: Mang dao nguoc (Worst Case) ---")
# Ví dụ 3: Mảng đảo ngược (So sánh tối đa, hoán đổi tối đa)
arr3 = [50, 40, 30, 20, 10] # N = 5
sorted_arr3, comp3, swap3 = selection_sort_analysis(arr3)
print(f"Mang da sap xep: {sorted_arr3}")
print(f"Tong so lan So sanh: {comp3}") # Vẫn là 10
print(f"Tong so lan Hoan doi: {swap3}") # N-1 = 4 hoán đổi thực tế
```

=> Kết quả sau khi chạy chương trình:

```
--- Vi du 1: Mang ngau nhien ---
Mang da sap xep: [11, 12, 22, 25, 64]
Tong so lan So sanh: 10
Tong so lan Hoan doi: 3


--- Vi du 2: Mang da sap xep (Best Case) ---
Mang da sap xep: [10, 20, 30, 40, 50]
Tong so lan So sanh: 10
Tong so lan Hoan doi: 0

--- Vi du 3: Mang dao nguoc (Worst Case) ---
Mang da sap xep: [10, 20, 30, 40, 50]
Tong so lan So sanh: 10
Tong so lan Hoan doi: 2
```

2.2. Code C++:

```cpp
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
using namespace std;         // Tiền tố std
tuple<vector<int>, int, int> selection_sort_analysis(vector<int> arr)
{
    // Thuc hien Selection Sort va dem chinh xac so lan so sanh va hoan doi.
    int n = arr.size();
    int comparison_count = 0;
    int swap_count = 0;
    // Vong lap ngoai: Duyet qua tung vi tri can sap xep (tu 0 den n-2)
    // i chay tu 0 den n-2 (tong cong n-1 lan)
    for (int i = 0; i < n - 1; i++)
    {
        // Gia su phan tu nho nhat la phan tu dau tien cua phan chua sap xep
        int min_idx = i;
        // Vong lap trong: Tim phan tu nho nhat trong phan chua sap xep
        // j chay tu i+1 den n-1
        for (int j = i + 1; j < n; j++)
```

```
        {
            // 1. Dem lan so sanh (So sanh arr[j] < arr[min_idx])
            comparison_count++;
            if (arr[j] < arr[min_idx])
            {
                min_idx = j; // Cap nhat chi muc cua phan tu nho nhat
            }
        }
        // 2. Hoan doi: Chi thuc hien 1 lan sau khi vong lap j ket thuc
        // Chi hoan doi neu phan tu nho nhat khong phai la arr[i]
        if (min_idx != i)
        {
            swap(arr[i], arr[min_idx]);
            swap_count++; // Dem lan hoan doi thuc te
        }
    }
    // Tra ve mang da sap xep va so lieu phan tich
    return make_tuple(arr, comparison_count, swap_count);
}
int main()
{
    // ----------------- CHAY THU -----------------
    // Vi du 1: Mang ngau nhien
    vector<int> arr1 = {64, 25, 12, 22, 11}; // N = 5
    auto [sorted_arr1, comp1, swap1] = selection_sort_analysis(arr1);
    cout << "--- Vi du 1: Mang ngau nhien ---" << endl;
    cout << "Mang da sap xep: ";
    for (int x : sorted_arr1) cout << x << " ";
    cout << endl;
    // Du doan so sanh: 5*(5-1)/2 = 10
    cout << "Tong so lan So sanh: " << comp1 << endl;
    cout << "Tong so lan Hoan doi: " << swap1 << endl; // 4 hoan doi thuc te
    cout << "\n--- Vi du 2: Mang da sap xep (Best Case) ---" << endl;
    // Vi du 2: Mang da sap xep (So sanh toi da, hoan doi toi thieu)
    vector<int> arr2 = {10, 20, 30, 40, 50}; // N = 5
    auto [sorted_arr2, comp2, swap2] = selection_sort_analysis(arr2);
    cout << "Mang da sap xep: ";
    for (int x : sorted_arr2) cout << x << " ";
    cout << endl;
    cout << "Tong so lan So sanh: " << comp2 << endl; // Van la 10 (luon la O(N^2))
    cout << "Tong so lan Hoan doi: " << swap2 << endl; // 0 hoan doi thuc te
    cout << "\n--- Vi du 3: Mang dao nguoc (Worst Case) ---" << endl;
    // Vi du 3: Mang dao nguoc (So sanh toi da, hoan doi toi da)
    vector<int> arr3 = {50, 40, 30, 20, 10}; // N = 5
    auto [sorted_arr3, comp3, swap3] = selection_sort_analysis(arr3);
    cout << "Mang da sap xep: ";
    for (int x : sorted_arr3) cout << x << " ";
    cout << endl;
    cout << "Tong so lan So sanh: " << comp3 << endl; // Van la 10
    cout << "Tong so lan Hoan doi: " << swap3 << endl; // N-1 = 4 hoan doi thuc te
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
--- Vi du 1: Mang ngau nhien ---
Mang da sap xep: 11 12 22 25 64
Tong so lan So sanh: 10
Tong so lan Hoan doi: 3
```

```
--- Vi du 2: Mang da sap xep (Best Case) ---
Mang da sap xep: 10 20 30 40 50
Tong so lan So sanh: 10
Tong so lan Hoan doi: 0

--- Vi du 3: Mang dao nguoc (Worst Case) ---
Mang da sap xep: 10 20 30 40 50
Tong so lan So sanh: 10
Tong so lan Hoan doi: 2
```

3. Bài tập 3 (Sắp xếp Giảm dần): Sửa đổi Bubble Sort để sắp xếp mảng theo thứ tự giảm dần.

3.1. Code Python:

```python
def bubble_sort_descending(arr):
    # Sắp xếp mảng theo thứ tự GIẢM DẦN (Descending Order) bằng Bubble Sort.
    n = len(arr)
    # Lặp qua tất cả các phần tử (n lần tối đa)
    for i in range(n):
        swapped = False
        # Vòng lặp trong: So sánh các cặp kề nhau
        # j chạy đến n - 1 - i vì i phần tử cuối cùng (nhỏ nhất) đã được sắp xếp
        for j in range(0, n - i - 1):
            # --- ĐIỀU KIỆN SẮP XẾP GIẢM DẦN ĐÃ SỬA ĐỔI ---
            # Hoán đổi nếu phần tử bên trái (arr[j]) NHỎ HƠN phần tử bên phải
(arr[j + 1])
            if arr[j] < arr[j + 1]:
                # Hoán đổi
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
                swapped = True
        # Tối ưu hóa: Nếu không có hoán đổi nào trong lần lặp này, mảng đã sắp xếp
        if not swapped:
            break
    return arr
# ----------------- CHẠY THỬ -----------------
arr1 = [64, 34, 25, 12, 22, 11, 90]
print(f"Mang goc: {arr1}")
print(f"Sap xep giam dan: {bubble_sort_descending(arr1)}")
# Kết quả: [90, 64, 34, 25, 22, 12, 11]
arr2 = [1, 5, 2, 8, 3]
print(f"Mang goc: {arr2}")
print(f"Sap xep giam dan: {bubble_sort_descending(arr2)}")
# Kết quả: [8, 5, 3, 2, 1]
```

=> Kết quả sau khi chạy chương trình:

```
Mang goc: [64, 34, 25, 12, 22, 11, 90]
Sap xep giam dan: [90, 64, 34, 25, 22, 12, 11]
Mang goc: [1, 5, 2, 8, 3]
Sap xep giam dan: [8, 5, 3, 2, 1]
```

3.2. Code C++:

```cpp
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
using namespace std;         // Tiền tố std
vector<int> bubble_sort_descending(vector<int> arr)
{
    // Sap xep mang theo thu tu GIAM DAN (Descending Order) bang Bubble Sort.
    int n = arr.size();
    // Lap qua tat ca cac phan tu (n lan toi da)
```

```
    for (int i = 0; i < n; i++)
    {
        bool swapped = false;
        // Vong lap trong: So sanh cac cap ke nhau
        // j chay den n - 1 - i vi i phan tu cuoi cung (nho nhat) da duoc sap xep
        for (int j = 0; j < n - i - 1; j++)
        {
            // --- DIEU KIEN SAP XEP GIAM DAN DA SUA DOI ---
            // Hoan doi neu phan tu ben trai (arr[j]) NHO HON phan tu ben phai
(arr[j + 1])
            if (arr[j] < arr[j + 1])
            {
                // Hoan doi
                swap(arr[j], arr[j + 1]);
                swapped = true;
            }
        }
        // Toi uu hoa: Neu khong co hoan doi nao trong lan lap nay, mang da sap xep
        if (!swapped)
        {
            break;
        }
    }
    return arr;
}
int main()
{
    // ----------------- CHAY THU -----------------
    vector<int> arr1 = {64, 34, 25, 12, 22, 11, 90};
    cout << "Mang goc: ";
    for (int x : arr1) cout << x << " ";
    cout << endl;
    vector<int> sorted1 = bubble_sort_descending(arr1);
    cout << "Sap xep giam dan: ";
    for (int x : sorted1) cout << x << " ";
    cout << endl;
    // Ket qua: [90, 64, 34, 25, 22, 12, 11]
    vector<int> arr2 = {1, 5, 2, 8, 3};
    cout << "Mang goc: ";
    for (int x : arr2) cout << x << " ";
    cout << endl;
    vector<int> sorted2 = bubble_sort_descending(arr2);
    cout << "Sap xep giam dan: ";
    for (int x : sorted2) cout << x << " ";
    cout << endl;
    // Ket qua: [8, 5, 3, 2, 1]
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
Mang goc: 64 34 25 12 22 11 90
Sap xep giam dan: 90 64 34 25 22 12 11
Mang goc: 1 5 2 8 3
Sap xep giam dan: 8 5 3 2 1
```

BÀI 20. THUẬT TOÁN SẮP XẾP NHANH (QUICK SORT)

I. Mục Tiêu:

- **Kiến thức:** HS hiểu rõ nguyên lý **Chia để Trị** và cơ chế hoạt động của QuickSort. Nắm vững vai trò và cách chọn **Phần tử Chốt (Pivot)**. Hiểu rõ hàm **Phân Hoạch (Partition)** là chìa khóa của thuật toán.
- **Kỹ năng:** HS cài đặt được hàm Phân hoạch O(N) và thuật toán QuickSort đệ quy hoàn chỉnh bằng Python và C++.
- **Tư duy:** HS phân tích được độ phức tạp thời gian tốt nhất (O(N log N)) và xấu nhất ($O(N^2)$) của QuickSort và cách lựa chọn pivot ảnh hưởng đến hiệu suất.

II. Phân Bổ Thời Gian (90 Phút):

Hoạt Động	Nội Dung	Thời gian (Phút)
Ôn tập	Ôn tập các thuật toán $O(N^2)$, Giới thiệu Chia để Trị	10
Lý thuyết 1	QuickSort: Nguyên lý và lựa chọn Pivot	15
Thực hành 1	Bài 1. Cài đặt Hàm Phân hoạch (Partition Scheme)	30
Lý thuyết 2	Cơ chế đệ quy và Độ phức tạp	10
Thực hành 2	Bài 2. Cài đặt QuickSort hoàn chỉnh	20
Tổng kết	So sánh QuickSort và $O(N^2)$, Bài tập về nhà	5
Tổng cộng		**90**

III. Lý Thuyết 1: QuickSort - Nguyên lý và Pivot (15 phút)

1. Nguyên lý Chia để Trị (Divide and Conquer):

QuickSort hoạt động theo ba bước chính:

- **Bước 1. Chia (Divide):** Chọn một phần tử làm **Chốt (Pivot)**.
- **Bước 2. Trị (Conquer) / Phân Hoạch (Partition):** Sắp xếp lại mảng sao cho tất cả các phần tử nhỏ hơn Pivot nằm ở bên trái và tất cả các phần tử lớn hơn Pivot nằm ở bên phải. Pivot nằm ở vị trí cuối cùng đúng của nó.
- **Bước 3. Hợp nhất (Combine):** Gọi QuickSort đệ quy cho hai mảng con (trái và phải của Pivot). (Bước hợp nhất trong QuickSort là rỗng vì việc phân hoạch đã tự động sắp xếp).

2. Lựa chọn Pivot:

Việc chọn Pivot ảnh hưởng lớn đến hiệu suất:

- **Tốt nhất:** Chọn Pivot luôn chia mảng thành hai nửa gần bằng nhau (Ví dụ: Trung vị). Độ phức tạp O(N log N).
- **Xấu nhất:** Chọn Pivot luôn là phần tử nhỏ nhất hoặc lớn nhất (Ví dụ: Mảng đã sắp xếp và luôn chọn phần tử đầu tiên). Độ phức tạp $O(N^2)$.
- **Thực tế:** Thường chọn phần tử đầu tiên, cuối cùng, hoặc phần tử ngẫu nhiên để đơn giản.

IV. Thực Hành 1: Cài đặt Hàm Phân hoạch (Partition) (30 phút)

- Chúng ta sẽ sử dụng Scheme Partition của **Lomuto** hoặc **Hoare**. Ta chọn Lomuto vì nó dễ hiểu và cài đặt.

- **Mục tiêu của hàm Partition:** Sắp xếp mảng con **arr[low..high]** xung quanh **Pivot (arr[high])** và trả về chỉ mục cuối cùng của **Pivot**.

1. Code Python:

```python
def partition(arr, low, high):
    # 1. Chọn Pivot là phần tử cuối cùng
    pivot = arr[high]
    i = low - 1  # Chỉ mục của ranh giới giữa khu vực < Pivot và > Pivot
    # 2. Vòng lặp phân hoạch: Duyệt qua các phần tử từ low đến high-1
    for j in range(low, high):
        # Nếu phần tử hiện tại nhỏ hơn hoặc bằng pivot
        if arr[j] <= pivot:
            i += 1
            # Hoán đổi arr[i] và arr[j] (Đưa phần tử nhỏ hơn/bằng vào vị trí i)
            arr[i], arr[j] = arr[j], arr[i]
    # 3. Đặt Pivot vào vị trí đúng (CHỈ THỰC HIỆN MỘT LẦN, SAU KHI FOR J KẾT THÚC)
    # Hoán đổi pivot (arr[high]) với phần tử lớn hơn đầu tiên (arr[i+1])
    arr[i + 1], arr[high] = arr[high], arr[i + 1]
    # 4. Trả về chỉ mục của pivot sau khi phân hoạch
    return i + 1
# Chạy thử
arr = [10, 80, 30, 90, 40, 50, 70]
pivot_index = partition(arr, 0, 6)
print(f"Chi muc Pivot: {pivot_index}")
print(f"Mang sau Partition: {arr}")
# Kết quả đúng: [10, 30, 40, 50, 70, 80, 90]
# (Pivot 70 nằm ở index 4, và các phần tử 80, 90 nằm bên phải nó)
```

=> Kết quả sau khi chạy chương trình:

```
Chi muc Pivot: 4
Mang sau Partition: [10, 30, 40, 50, 70, 90, 80]
```

- Giải thích Code Python:

Chỉ mục i: i theo dõi ranh giới giữa các phần tử nhỏ hơn Pivot và lớn hơn Pivot. Bất cứ khi nào tìm thấy một phần tử nhỏ hơn, ta tăng i và hoán đổi nó vào vị trí arr[i].

2. Code C++:

```cpp
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
using namespace std;         // Tiền tố std
int partition(vector<int>& arr, int low, int high)
{
    // 1. Chon Pivot la phan tu cuoi cung
    int pivot = arr[high];
    int i = low - 1;  // Chi muc cua ranh gioi giua khu vuc < Pivot va > Pivot
    // 2. Vong lap phan hoach: Duyet qua cac phan tu tu low den high-1
    for (int j = low; j < high; j++)
    {
        // Neu phan tu hien tai nho hon hoac bang pivot
        if (arr[j] <= pivot)
        {
            i++;
            // Hoan doi arr[i] va arr[j] (Dua phan tu nho hon/bang vao vi tri i)
            swap(arr[i], arr[j]);
        }
    }
    // 3. Dat Pivot vao vi tri dung (CHI THUC HIEN MOT LAN, SAU KHI FOR J KET THUC)
    // Hoan doi pivot (arr[high]) voi phan tu lon hon dau tien (arr[i+1])
    swap(arr[i + 1], arr[high]);
```

```
    // 4. Tra ve chi muc cua pivot sau khi phan hoach
    return i + 1;
}
int main()
{
    // Chay thu
    vector<int> arr = {10, 80, 30, 90, 40, 50, 70};
    int pivot_index = partition(arr, 0, 6);
    cout << "Chi muc Pivot: " << pivot_index << endl;
    cout << "Mang sau Partition: ";
    for (int x : arr)
    {
        cout << x << " ";
    }
    cout << endl;
    // Ket qua dung: [10, 30, 40, 50, 70, 80, 90]
    // (Pivot 70 nam o index 4, va cac phan tu 80, 90 nam ben phai no)
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
Chi muc Pivot: 4
Mang sau Partition: [10, 30, 40, 50, 70, 90, 80]
```

- Giải thích Code C++:

Kiểu trả về: Hàm partition trả về **int (chỉ mục Pivot)**. Cần sử dụng **swap** để hoán đổi. Logic của vòng lặp **for j** và chỉ mục **i** hoàn toàn tương tự Python.

V. Lý thuyết 2: Cơ chế đệ quy và Độ phức tạp (10 phút)

1. Cơ chế đệ quy:

- Sau khi hàm **partition(arr, low, high)** trả về chỉ mục **Pivot (P)**, ta gọi lại chính hàm **QuickSort** cho hai mảng con:

  + QuickSort(arr, low, P-1) => Mảng con bên trái (nhỏ hơn Pivot).

  + QuickSort(arr, P+1, high) => Mảng con bên phải (lớn hơn Pivot).

- Trường hợp cơ sở (Base Case) cho đệ quy là khi low ≥ high (mảng con chỉ có 0 hoặc 1 phần tử).

2. Độ phức tạp (Complexity):

Trường hợp	Độ phức tạp	Mô tả
Tốt nhất / Trung bình	O(N log N)	Khi Pivot luôn chia mảng thành hai nửa gần bằng nhau.
Xấu nhất	$O(N^2)$	Khi Pivot luôn là phần tử nhỏ nhất hoặc lớn nhất (Ví dụ: Mảng đã sắp xếp hoặc đảo ngược).
Không gian	O(log N)	Sử dụng bộ nhớ cho các lệnh gọi đệ quy (Stack Call), trung bình là log N.

VI. Thực Hành 2: Cài đặt QuickSort hoàn chỉnh (20 phút)

1. Bài 2: Cài đặt QuickSort hoàn chỉnh (Sử dụng hàm Partition đã viết)

1.1. Code Python:

```
# Sử dụng hàm partition() đã viết ở trên
def partition(arr, low, high):
    # 1. Chọn Pivot là phần tử cuối cùng
    pivot = arr[high]
    i = low - 1  # Chỉ mục của ranh giới giữa khu vực < Pivot và > Pivot
```

```
    # 2. Vòng lặp phân hoạch: Duyệt qua các phần tử từ low đến high-1
    for j in range(low, high):
        # Nếu phần tử hiện tại nhỏ hơn hoặc bằng pivot
        if arr[j] <= pivot:
            i += 1
            # Hoán đổi arr[i] và arr[j] (Đưa phần tử nhỏ hơn/bằng vào vị trí i)
            arr[i], arr[j] = arr[j], arr[i]
    # 3. Đặt Pivot vào vị trí đúng (CHỈ THỰC HIỆN MỘT LẦN, SAU KHI FOR J KẾT THÚC)
    # Hoán đổi pivot (arr[high]) với phần tử lớn hơn đầu tiên (arr[i+1])
    arr[i + 1], arr[high] = arr[high], arr[i + 1]
    # 4. Trả về chỉ mục của pivot sau khi phân hoạch
    return i + 1
# Sắp xếp
def quick_sort(arr, low, high):
    # Trường hợp cơ sở (Base Case): Dừng đệ quy
    if low < high:
        # 1. Phân hoạch mảng và lấy chỉ mục pivot
        pivot_index = partition(arr, low, high)
        # 2. Gọi đệ quy cho mảng con bên trái
        quick_sort(arr, low, pivot_index - 1)
        # 3. Gọi đệ quy cho mảng con bên phải
        quick_sort(arr, pivot_index + 1, high)
    # Trả về mảng đã sắp xếp (không cần return trong đệ quy In-place)
# Chạy thử
arr = [10, 80, 30, 90, 40, 50, 70]
n = len(arr)
quick_sort(arr, 0, n - 1)
print(arr) # [10, 30, 40, 50, 70, 80, 90]
```

=> Kết quả sau khi chạy chương trình:

```
[10, 30, 40, 50, 70, 80, 90]
```

- Giải thích Code Python:

Truyền tham số: Cần truyền **low** và **high** để xác định phạm vi của mảng con đang được sắp xếp. Python truyền danh sách dưới dạng tham chiếu nên việc sắp xếp là *In-place*.

1.2. Code C++:

```
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
using namespace std;         // Tiền tố std
// Su dung ham partition() da viet o tren
int partition(vector<int>& arr, int low, int high)
{
    // 1. Chon Pivot la phan tu cuoi cung
    int pivot = arr[high];
    int i = low - 1;  // Chi muc cua ranh gioi giua khu vuc < Pivot va > Pivot
    // 2. Vong lap phan hoach: Duyet qua cac phan tu tu low den high-1
    for (int j = low; j < high; j++)
    {
        // Neu phan tu hien tai nho hon hoac bang pivot
        if (arr[j] <= pivot)
        {
            i++;
            // Hoan doi arr[i] va arr[j] (Dua phan tu nho hon/bang vao vi tri i)
            swap(arr[i], arr[j]);
        }
    }
    // 3. Dat Pivot vao vi tri dung (CHI THUC HIEN MOT LAN, SAU KHI FOR J KET THUC)
```

```
    // Hoan doi pivot (arr[high]) voi phan tu lon hon dau tien (arr[i+1])
    swap(arr[i + 1], arr[high]);
    // 4. Tra ve chi muc cua pivot sau khi phan hoach
    return i + 1;
}
// Sap xep
void quick_sort(vector<int>& arr, int low, int high)
{
    // Truong hop co so (Base Case): Dung de quy
    if (low < high)
    {
        // 1. Phan hoach mang va lay chi muc pivot
        int pivot_index = partition(arr, low, high);
        // 2. Goi de quy cho mang con ben trai
        quick_sort(arr, low, pivot_index - 1);
        // 3. Goi de quy cho mang con ben phai
        quick_sort(arr, pivot_index + 1, high);
    }
    // Tra ve mang da sap xep (khong can return trong de quy In-place)
}
int main()
{
    // Chay thu
    vector<int> arr = {10, 80, 30, 90, 40, 50, 70};
    int n = arr.size();
    quick_sort(arr, 0, n - 1);
    for (int x : arr)
    {
        cout << x << " ";
    }
    cout << endl;
    // Ket qua: [10, 30, 40, 50, 70, 80, 90]
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
10 30 40 50 70 80 90
```

- Giải thích Code C++:

Tham chiếu: C++ sử dụng **vector<int>& arr** để truyền mảng dưới dạng tham chiếu, đảm bảo việc sắp xếp được thực hiện trực tiếp trên mảng gốc (In-place).

VII. Bài Tập Về Nhà & Tổng Kết (5 phút)

1. Bài tập 1 (Phân tích Worst Case): Chạy QuickSort với dữ liệu đầu vào là một mảng đã sắp xếp [1, 2, 3, 4, 5] và luôn chọn Pivot là phần tử cuối cùng. Quan sát số lần gọi đệ quy.

1.1. Code Python:

```
# Biến toàn cục để theo dõi số lần gọi đệ quy
recursion_count = 0
def partition(arr, low, high):
    # Thực hiện bước Phân hoạch Lomuto (Pivot là phần tử cuối cùng).
    pivot = arr[high]
    i = low - 1
    for j in range(low, high):
        if arr[j] <= pivot:
            i += 1
            arr[i], arr[j] = arr[j], arr[i]
    arr[i + 1], arr[high] = arr[high], arr[i + 1]
```

```
    return i + 1
def quick_sort_recursive(arr, low, high):
    # Hàm QuickSort đệ quy, được sửa đổi để đếm số lần gọi.
    global recursion_count
    # Tăng bộ đếm mỗi khi hàm được gọi
    recursion_count += 1
    # Trường hợp cơ sở: Dừng đệ quy
    if low < high:
        # 1. Phân hoạch mảng
        pivot_index = partition(arr, low, high)
        # 2. Gọi đệ quy cho mảng con bên trái
        quick_sort_recursive(arr, low, pivot_index - 1)
        # 3. Gọi đệ quy cho mảng con bên phải
        quick_sort_recursive(arr, pivot_index + 1, high)
# ----------------- CHẠY THỬ TRƯỜNG HỢP XẤU NHẤT -----------------
# Dữ liệu đầu vào: Mảng đã sắp xếp (Worst Case khi Pivot là cuối cùng)
arr_worst = [1, 2, 3, 4, 5]
N = len(arr_worst)
recursion_count = 0 # Đặt lại bộ đếm
print(f"Mang dau vao: {arr_worst}")
print(f"Kich thuoc mang (N): {N}")
# Gọi QuickSort
quick_sort_recursive(arr_worst, 0, N - 1)
# Tổng số lần gọi đệ quy (bao gồm cả lần gọi ban đầu)
print("-" * 40)
print(f"Mang sau sap xep: {arr_worst}")
print(f"Tong so lan goi ham QuickSort: {recursion_count}")

# ----------------- PHÂN TÍCH KẾT QUẢ -----------------
# Với N=5, thuật toán sẽ đi theo chuỗi gọi sau:
# quick_sort(0, 4) -> (Pivot 5) -> P=4. Gọi con (0, 3) và (5, 4 [rỗng])
# quick_sort(0, 3) -> (Pivot 4) -> P=3. Gọi con (0, 2) và (4, 3 [rỗng])
# quick_sort(0, 2) -> (Pivot 3) -> P=2. Gọi con (0, 1) và (3, 2 [rỗng])
# quick_sort(0, 1) -> (Pivot 2) -> P=1. Gọi con (0, 0) và (2, 1 [rỗng])
# quick_sort(0, 0) -> (Pivot 1) -> P=0. Gọi con (-1, 0 [rỗng]) và (1, 0 [rỗng])

# Số lần gọi đệ quy (khi low <= high):
# (0, 4), (0, 3), (0, 2), (0, 1), (0, 0) -> 5 lần
# Calls when low > high (mảng rỗng): 5 + 4 + 3 + 2 + 1 = 15 lần
# Tổng cộng: 5 + 15 = 20 lần gọi đệ quy.
# Hoặc N*2 - 1 lần gọi (N=5): 5 + (5-1) + (5-1) + ... = 10 + 5 + 4 = 21

# Kết quả đếm thực tế (recursion_count) sẽ cho biết tổng số lần hàm được gọi
# (bao gồm các lần gọi cho mảng rỗng)
```

=> Kết quả sau khi chạy chương trình:

```
Mang dau vao: [1, 2, 3, 4, 5]
Kich thuoc mang (N): 5
----------------------------------
Mang sau sap xep: [1, 2, 3, 4, 5]
Tong so lan goi ham QuickSort: 9
```

1.2. Code C++:

```
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
using namespace std;         // Tiền tố std
// Bien toan cuc de theo doi so lan goi de quy
int recursion_count = 0;
int partition(vector<int>& arr, int low, int high)
```

```
{
    // Thuc hien buoc Phan hoach Lomuto (Pivot la phan tu cuoi cung).
    int pivot = arr[high];
    int i = low - 1;
    for (int j = low; j < high; j++)
    {
        if (arr[j] <= pivot)
        {
            i++;
            swap(arr[i], arr[j]);
        }
    }
    swap(arr[i + 1], arr[high]);
    return i + 1;
}
void quick_sort_recursive(vector<int>& arr, int low, int high)
{
    // Ham QuickSort de quy, duoc sua doi de dem so lan goi.
    // Tang bo dem moi khi ham duoc goi
    recursion_count++;
    // Truong hop co so: Dung de quy
    if (low < high)
    {
        // 1. Phan hoach mang
        int pivot_index = partition(arr, low, high);
        // 2. Goi de quy cho mang con ben trai
        quick_sort_recursive(arr, low, pivot_index - 1);
        // 3. Goi de quy cho mang con ben phai
        quick_sort_recursive(arr, pivot_index + 1, high);
    }
}
int main()
{
    // ----------------- CHAY THU TRUONG HOP XAU NHAT -----------------
    // Du lieu dau vao: Mang da sap xep (Worst Case khi Pivot la cuoi cung)
    vector<int> arr_worst = {1, 2, 3, 4, 5};
    int N = arr_worst.size();
    recursion_count = 0; // Dat lai bo dem
    cout << "Mang dau vao: ";
    for (int x : arr_worst) cout << x << " ";
    cout << endl;
    cout << "Kich thuoc mang (N): " << N << endl;
    // Goi QuickSort
    quick_sort_recursive(arr_worst, 0, N - 1);
    cout << string(40, '-') << endl;
    cout << "Mang sau sap xep: ";
    for (int x : arr_worst) cout << x << " ";
    cout << endl;
    // Tong so lan goi de quy (bao gom ca lan goi ban dau)
    cout << "Tong so lan goi ham QuickSort: " << recursion_count << endl;

    /*
    ----------------- PHAN TICH KET QUA -----------------

    Voi N=5, thuat toan se di theo chuoi goi sau:
    quick_sort(0, 4) -> (Pivot 5) -> P=4. Goi con (0, 3) va (5, 4 [rong])
    quick_sort(0, 3) -> (Pivot 4) -> P=3. Goi con (0, 2) va (4, 3 [rong])
    quick_sort(0, 2) -> (Pivot 3) -> P=2. Goi con (0, 1) va (3, 2 [rong])
```

```
    quick_sort(0, 1) -> (Pivot 2) -> P=1. Goi con (0, 0) va (2, 1 [rong])
    quick_sort(0, 0) -> (Pivot 1) -> P=0. Goi con (-1, 0 [rong]) va (1, 0 [rong])

    So lan goi de quy (khi low <= high):
    (0, 4), (0, 3), (0, 2), (0, 1), (0, 0) -> 5 lan

    Cac lan goi voi mang rong (low > high) van duoc dem
    Tong so lan goi se duoc the hien boi bien recursion_count
    */

    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
Mang dau vao: 1 2 3 4 5
Kich thuoc mang (N): 5
----------------------------------------
Mang sau sap xep: 1 2 3 4 5
Tong so lan goi ham QuickSort: 9
```

2. Bài tập 2 (Tìm kiếm Nhị phân): Cài đặt hàm Tìm kiếm Nhị phân (Binary Search) và phân tích độ phức tạp O(log N).

2.1. Code Python:

```python
def binary_search(arr, target):
    # Thực hiện tìm kiếm Nhị phân (Iterative) trên mảng đã sắp xếp.
    # Tham số:
    #   arr (list): Danh sách các phần tử đã sắp xếp.
    #   target (any): Giá trị cần tìm kiếm.
    # Trả về:
    # int: Chỉ mục (index) của phần tử target đầu tiên tìm thấy.
    #   Trả về -1 nếu không tìm thấy.
    # Khởi tạo các chỉ mục ranh giới
    low = 0
    high = len(arr) - 1
    # Tiếp tục tìm kiếm khi ranh giới dưới <= ranh giới trên
    while low <= high:
        # Tính chỉ mục giữa (mid)
        # Sử dụng (low + high) // 2 để tránh lỗi tràn số trong các ngôn ngữ khác,
        # nhưng trong Python thì không cần lo lắng nhiều.
        mid = (low + high) // 2
        # So sánh target với phần tử tại mid
        if arr[mid] == target:
            # Trường hợp 1: Tìm thấy mục tiêu
            return mid
        elif arr[mid] < target:
            # Trường hợp 2: Mục tiêu lớn hơn mid.
            # Bỏ qua nửa trái, dịch chuyển ranh giới dưới lên mid + 1
            low = mid + 1
        else: # arr[mid] > target
            # Trường hợp 3: Mục tiêu nhỏ hơn mid.
            # Bỏ qua nửa phải, dịch chuyển ranh giới trên xuống mid - 1
            high = mid - 1
    # Nếu vòng lặp kết thúc mà không tìm thấy
    return -1
# ----------------- CHẠY THỬ -----------------
data = [5, 12, 18, 20, 25, 30, 45, 50, 75, 80] # ĐÃ SẮP XẾP
target1 = 30
target2 = 70
```

```
print(f"Du lieu da sap xep: {data}")
print("-" * 40)
# Trường hợp 1: Tìm thấy
index1 = binary_search(data, target1)
print(f"Tim kiem {target1}: Index = {index1}") # Kết quả: 5
# Trường hợp 2: Không tìm thấy
index2 = binary_search(data, target2)
print(f"Tim kiem {target2}: Index = {index2}") # Kết quả: -1
```

=> Kết quả sau khi chạy chương trình:

```
Du lieu da sap xep: [5, 12, 18, 20, 25, 30, 45, 50, 75, 80]
----------------------------------------
Tim kiem 30: Index = 5
Tim kiem 70: Index = -1
```

2.2. Code C++:

```
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
using namespace std;         // Tiền tố std
int binary_search(vector<int>& arr, int target)
{
    // Thuc hien tim kiem Nhi phan (Iterative) tren mang da sap xep.
    // Tham so:
    //   arr (vector): Danh sach cac phan tu da sap xep.
    //   target (int): Gia tri can tim kiem.
    // Tra ve:
    // int: Chi muc (index) cua phan tu target dau tien tim thay.
    //   Tra ve -1 neu khong tim thay.
    // Khoi tao cac chi muc ranh gioi
    int low = 0;
    int high = arr.size() - 1;
    // Tiep tuc tim kiem khi ranh gioi duoi <= ranh gioi tren
    while (low <= high)
    {
        // Tinh chi muc giua (mid)
        // Su dung (low + high) / 2 de tranh loi tran so
        int mid = (low + high) / 2;
        // So sanh target voi phan tu tai mid
        if (arr[mid] == target)
        {
            // Truong hop 1: Tim thay muc tieu
            return mid;
        }
        else if (arr[mid] < target)
        {
            // Truong hop 2: Muc tieu lon hon mid.
            // Bo qua nua trai, dich chuyen ranh gioi duoi len mid + 1
            low = mid + 1;
        }
        else    // arr[mid] > target
        {
            // Truong hop 3: Muc tieu nho hon mid.
            // Bo qua nua phai, dich chuyen ranh gioi tren xuong mid - 1
            high = mid - 1;
        }
    }
    // Neu vong lap ket thuc ma khong tim thay
    return -1;
}
```

```
int main()
{
    // ----------------- CHAY THU -----------------
    vector<int> data = {5, 12, 18, 20, 25, 30, 45, 50, 75, 80}; // DA SAP XEP
    int target1 = 30;
    int target2 = 70;
    cout << "Du lieu da sap xep: ";
    for (int x : data) cout << x << " ";
    cout << endl;
    cout << string(40, '-') << endl;
    // Truong hop 1: Tim thay
    int index1 = binary_search(data, target1);
    cout << "Tim kiem " << target1 << ": Index = " << index1 << endl;
    // Ket qua: 5
    // Truong hop 2: Khong tim thay
    int index2 = binary_search(data, target2);
    cout << "Tim kiem " << target2 << ": Index = " << index2 << endl;
    // Ket qua: -1
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
Du lieu da sap xep: 5 12 18 20 25 30 45 50 75 80
----------------------------------------
Tim kiem 30: Index = 5
Tim kiem 70: Index = -1
```

2.3. Phân tích độ phức tạp O(log N):

- Tìm kiếm Nhị phân dựa trên nguyên lý chia không gian tìm kiếm thành hai nửa trong mỗi bước.

Bước lặp	Kích thước mảng
Ban đầu	N
1	N / 2
2	N / 4
k	$N / 2^k$

- Thuật toán kết thúc khi kích thước mảng con chỉ còn 1: **$N / 2^k$ gần bằng 1**
- Ta có phương trình: **$N = 2^k$**
- Để tìm số bước lặp **k**: $k = \log_2 N$
- **Kết luận:**

+ Độ phức tạp thời gian của Tìm kiếm Nhị phân là **O(log N)** (Big O of log N). Điều này có nghĩa là, khi kích thước mảng N tăng gấp đôi, số lần so sánh (bước lặp) chỉ tăng thêm 1. Ví dụ:

$N = 1000$: $\log_2(1000)$ tương đương 10 bước

$N = 1.000.000$: $\log_2(1.000.000)$ tương đương 20 bước

+ Đây là một thuật toán cực kỳ hiệu quả về mặt thời gian cho việc tìm kiếm trên dữ liệu đã được sắp xếp.

3. Bài tập 3 (Sửa Pivot): Sửa đổi hàm **partition** để lựa chọn Pivot là phần tử **ngẫu nhiên** thay vì phần tử cuối cùng để giảm thiểu khả năng xảy ra trường hợp xấu nhất.

3.1. Code Python:

```
import random   # Cần thư viện random để tạo ra số ngẫu nhiên
def partition_random(arr, low, high):
```

```
    # 1. Chọn ngẫu nhiên một phần tử làm Pivot.
    # 2. Hoán đổi Pivot ngẫu nhiên đó với phần tử cuối cùng (arr[high]).
    # 3. Thực hiện phân hoạch Lomuto như bình thường.
    # --- BƯỚC 1: Chọn Pivot Ngẫu nhiên ---
    # Sinh một chỉ mục ngẫu nhiên trong phạm vi [low, high]
    rand_idx = random.randint(low, high)
    # --- BƯỚC 2: Hoán đổi Pivot Ngẫu nhiên với arr[high] ---
    # Đưa Pivot ngẫu nhiên về vị trí cuối cùng để áp dụng logic Lomuto
    arr[rand_idx], arr[high] = arr[high], arr[rand_idx]
    # Pivot hiện tại là phần tử cuối cùng (arr[high])
    pivot = arr[high]
    i = low - 1  # Chỉ mục của ranh giới khu vực <= Pivot
    # 3. Vòng lặp phân hoạch (Partition Lomuto)
    for j in range(low, high):
        if arr[j] <= pivot:
            i += 1
            # Hoán đổi arr[i] và arr[j]
            arr[i], arr[j] = arr[j], arr[i]
    # 4. Đặt Pivot vào vị trí đúng (sau khi for j kết thúc)
    arr[i + 1], arr[high] = arr[high], arr[i + 1]
    # 5. Trả về chỉ mục Pivot mới
    return i + 1
def quick_sort_randomized(arr, low, high):
    # Hàm QuickSort sử dụng Random Pivot Selection.
    if low < high:
        # Sử dụng hàm partition mới
        pivot_index = partition_random(arr, low, high)
        # Gọi đệ quy cho trái và phải
        quick_sort_randomized(arr, low, pivot_index - 1)
        quick_sort_randomized(arr, pivot_index + 1, high)
# ----------------- CHẠY THỬ TRƯỜNG HỢP XẤU NHẤT -----------------
# Dữ liệu: Mảng đã sắp xếp (Worst Case khi Pivot là cuối cùng,
# nhưng Randomized Pivot sẽ tránh được)
arr_worst = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
N = len(arr_worst)
print(f"Mang dau vao (Worst Case): {arr_worst}")
# Gọi QuickSort Randomized
quick_sort_randomized(arr_worst, 0, N - 1)
print(f"Mang sau sap xep ngau nhien: {arr_worst}")
```

=> Kết quả sau khi chạy chương trình:

```
Mang dau vao (Worst Case): [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Mang sau sap xep ngau nhien: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

3.2. Code C++:

```
#include <iostream>         // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>    // Thư viện tổng hợp
using namespace std;        // Tiền tố std
int partition_random(vector<int>& arr, int low, int high)
{
    // 1. Chon ngau nhien mot phan tu lam Pivot.
    // 2. Hoan doi Pivot ngau nhien do voi phan tu cuoi cung (arr[high]).
    // 3. Thuc hien phan hoach Lomuto nhu binh thuong.
    // --- BUOC 1: Chon Pivot Ngau nhien ---
    // Sinh mot chi muc ngau nhien trong pham vi [low, high]
    int rand_idx = low + rand() % (high - low + 1);
    // --- BUOC 2: Hoan doi Pivot Ngau nhien voi arr[high] ---
    // Dua Pivot ngau nhien ve vi tri cuoi cung de ap dung logic Lomuto
    swap(arr[rand_idx], arr[high]);
```

```
    // Pivot hien tai la phan tu cuoi cung (arr[high])
    int pivot = arr[high];
    int i = low - 1;  // Chi muc cua ranh gioi khu vuc <= Pivot
    // 3. Vong lap phan hoach (Partition Lomuto)
    for (int j = low; j < high; j++)
    {
        if (arr[j] <= pivot)
        {
            i++;
            // Hoan doi arr[i] va arr[j]
            swap(arr[i], arr[j]);
        }
    }
    // 4. Dat Pivot vao vi tri dung (sau khi for j ket thuc)
    swap(arr[i + 1], arr[high]);
    // 5. Tra ve chi muc Pivot moi
    return i + 1;
}
void quick_sort_randomized(vector<int>& arr, int low, int high)
{
    // Ham QuickSort su dung Random Pivot Selection.
    if (low < high)
    {
        // Su dung ham partition moi
        int pivot_index = partition_random(arr, low, high);
        // Goi de quy cho trai va phai
        quick_sort_randomized(arr, low, pivot_index - 1);
        quick_sort_randomized(arr, pivot_index + 1, high);
    }
}
int main()
{
    // Khoi tao bo sinh so ngau nhien
    srand(time(NULL));
    // ----------------- CHAY THU TRUONG HOP XAU NHAT -----------------
    // Du lieu: Mang da sap xep (Worst Case khi Pivot la cuoi cung,
    // nhung Randomized Pivot se tranh duoc)
    vector<int> arr_worst = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    int N = arr_worst.size();
    cout << "Mang dau vao (Worst Case): ";
    for (int x : arr_worst) cout << x << " ";
    cout << endl;
    // Goi QuickSort Randomized
    quick_sort_randomized(arr_worst, 0, N - 1);
    cout << "Mang sau sap xep ngau nhien: ";
    for (int x : arr_worst) cout << x << " ";
    cout << endl;
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
Mang dau vao (Worst Case): 1 2 3 4 5 6 7 8 9 10
Mang sau sap xep ngau nhien: 1 2 3 4 5 6 7 8 9 10
```

BÀI 21. THUẬT TOÁN TÌM KIẾM NHỊ PHÂN

I. Mục Tiêu:

- **Kiến thức:** HS hiểu rõ nguyên tắc **Chia để Trị** áp dụng trong Binary Search và điều kiện tiên quyết (mảng phải được sắp xếp). Nắm vững độ phức tạp thời gian O(log N).
- **Kỹ năng:** HS cài đặt được Binary Search ở cả hai dạng: **Lặp** (Iterative) và **Đệ quy** (Recursive) bằng Python và C++. HS biết cách áp dụng Binary Search để giải các bài toán tìm kiếm nâng cao.
- **Tư duy:** HS so sánh được hiệu suất giữa Binary Search và Linear Search. Nhận biết khi nào nên ưu tiên sử dụng Binary Search.

II. Phân Bổ Thời Gian (90 Phút):

Hoạt Động	Nội Dung	Thời gian (Phút)
Ôn tập	Ôn tập QuickSort, giới thiệu O(log N)	10
Lý thuyết 1	Binary Search: Nguyên lý, Điều kiện và Độ phức tạp	15
Thực hành 1	Bài 1. Cài đặt Binary Search (Dạng Lặp)	25
Thực hành 2	Bài 2. Cài đặt Binary Search (Dạng Đệ quy)	25
Tổng kết	So sánh 2 dạng, Bài tập về nhà	15
Tổng cộng		**90**

III. Lý Thuyết 1: Binary Search - Nguyên lý và Độ phức tạp (15 phút)

1. Nguyên lý Hoạt động: Binary Search (Tìm kiếm Nhị phân) là một thuật toán tìm kiếm dựa trên nguyên lý **Chia để Trị (Divide and Conquer)**.

- **Điều kiện:** Mảng phải **BẮT BUỘC** được sắp xếp (tăng dần hoặc giảm dần).
- **So sánh:** So sánh giá trị mục tiêu (**target**) với phần tử ở **giữa** mảng (**mid**).
- **Chia đôi:**
  + Nếu **target** bằng **mid**, tìm kiếm thành công.
  + Nếu **target** nhỏ hơn **mid**, tiếp tục tìm kiếm ở **nửa trái** của mảng.
  + Nếu **target** lớn hơn **mid**, tiếp tục tìm kiếm ở **nửa phải** của mảng.
- Lặp lại quá trình cho đến khi tìm thấy hoặc mảng con không còn phần tử nào.

2. Độ phức tạp Thời gian:

- **Tốt nhất (Best Case):** O(1) (Phần tử mục tiêu nằm ngay chính giữa).
- **Trung bình/Xấu nhất (Worst Case):** O(log N). Vì thuật toán loại bỏ một nửa không gian tìm kiếm sau mỗi lần so sánh, số lần lặp tối đa là $\log_2 N$.
- **Không gian:**
  + O(1) (Dạng Lặp): Sử dụng bộ nhớ hằng số.
  + O(log N) (Dạng Đệ quy): Sử dụng bộ nhớ cho Stack Call.

IV. Thực Hành 1: Cài đặt Binary Search (Dạng Lặp) (25 phút)

Dạng lặp (Iterative) được ưu tiên vì hiệu quả bộ nhớ O(1).

1. Code Python:

```python
def binary_search_iterative(arr, target):
    low = 0
    high = len(arr) - 1
    # Lặp cho đến khi ranh giới dưới vượt qua ranh giới trên (low > high)
    while low <= high:
        # Tính chỉ mục giữa
        mid = (low + high) // 2
        # 1. Tìm thấy
        if arr[mid] == target:
            return mid
        # 2. Bỏ qua nửa trái (Target lớn hơn, tìm kiếm ở nửa phải)
        elif arr[mid] < target:
            low = mid + 1
        # 3. Bỏ qua nửa phải (Target nhỏ hơn, tìm kiếm ở nửa trái)
        else: # arr[mid] > target
            high = mid - 1
    # --- VỊ TRÍ ĐÚNG ---
    # Nếu vòng lặp kết thúc mà không tìm thấy (low > high), trả về -1
    return -1
# Chạy thử
arr = [2, 5, 8, 12, 16, 23, 38, 56, 72, 91]
print(binary_search_iterative(arr, 23)) # Kết quả: 5 (Đúng)
```

=> Kết quả sau khi chạy chương trình: 5

- Giải thích Code Python:

+ Vòng lặp while: Điều kiện **low <= high** đảm bảo vòng lặp bao gồm cả trường hợp **low = high** (còn 1 phần tử).

+ Cập nhật ranh giới: Luôn là **mid + 1** hoặc **mid - 1** để loại bỏ mid đã kiểm tra.

2. Code C++:

```cpp
#include <iostream>         // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>    // Thư viện tổng hợp
using namespace std;        // Tiền tố std
int binary_search_iterative(vector<int>& arr, int target)
{
    int low = 0;
    int high = arr.size() - 1;
    // Lap cho den khi ranh gioi duoi vuot qua ranh gioi tren (low > high)
    while (low <= high)
    {
        // Tinh chi muc giua
        int mid = (low + high) / 2;
        // 1. Tim thay
        if (arr[mid] == target)
        {
            return mid;
        }
        // 2. Bo qua nua trai (Target lon hon, tim kiem o nua phai)
        else if (arr[mid] < target)
        {
            low = mid + 1;
        }
        // 3. Bo qua nua phai (Target nho hon, tim kiem o nua trai)
        else    // arr[mid] > target
        {
```

```
            high = mid - 1;
        }
    }
    // --- VI TRI DUNG ---
    // Neu vong lap ket thuc ma khong tim thay (low > high), tra ve -1
    return -1;
}
int main()
{
    // Chay thu
    vector<int> arr = {2, 5, 8, 12, 16, 23, 38, 56, 72, 91};
    cout << binary_search_iterative(arr, 23) << endl;
    // Ket qua: 5 (Dung)
    return 0;
}
```

=> Kết quả sau khi chạy chương trình: 5

- Giải thích Code C++:

Tránh Tràn số: Trong C++, công thức **mid = low + (high - low) / 2** là một best practice để tránh trường hợp **low + high** vượt quá giới hạn của kiểu **int** khi **low** và **high** đều lớn.

V. Thực Hành 2: Cài đặt Binary Search (Dạng Đệ quy) (25 phút)

Dạng đệ quy trực quan hơn về mặt nguyên lý "Chia để Trị".

1. Code Python:

```
def binary_search_recursive(arr, target, low, high):
    # Thực hiện tìm kiếm Nhị phân bằng phương pháp Đệ quy.
    # 1. Trường hợp cơ sở (Base Case): Không tìm thấy
    # Khi ranh giới dưới vượt qua ranh giới trên, phạm vi tìm kiếm rỗng
    if low > high:
        return -1
    # Tính chỉ mục giữa
    mid = (low + high) // 2
    # 2. Trường hợp tìm thấy
    if arr[mid] == target:
        return mid
    # 3. Target nằm ở nửa trái (low đến mid - 1)
    # Nếu phần tử giữa lớn hơn target, ta biết target phải nằm bên trái (nếu có)
    elif arr[mid] > target:
        # Gọi đệ quy với ranh giới mới: high = mid - 1
        return binary_search_recursive(arr, target, low, mid - 1)
    # 4. Target nằm ở nửa phải (mid + 1 đến high)
    # Nếu phần tử giữa nhỏ hơn target, ta biết target phải nằm bên phải (nếu có)
    else:   # arr[mid] < target
        # Gọi đệ quy với ranh giới mới: low = mid + 1
        return binary_search_recursive(arr, target, mid + 1, high)
# ----------------- CHẠY THỬ -----------------
arr = [2, 5, 8, 12, 16, 23, 38, 56, 72, 91]
target1 = 72
target2 = 100
n = len(arr)
print(f"Du lieu da sap xep: {arr}")
print("-" * 40)
# Chạy thử 1: Tìm thấy
index1 = binary_search_recursive(arr, target1, 0, n - 1)
print(f"Tim kiem {target1}: Index = {index1}")
# Chạy thử 2: Không tìm thấy
```

```
index2 = binary_search_recursive(arr, target2, 0, n - 1)
print(f"Tim kiem {target2}: Index = {index2}")
```

=> Kết quả sau khi chạy chương trình:

```
Du lieu da sap xep: [2, 5, 8, 12, 16, 23, 38, 56, 72, 91]
----------------------------------------
Tim kiem 72: Index = 8
Tim kiem 100: Index = -1
```

- Giải thích Code Python:

+ Trường hợp cơ sở: Dừng đệ quy khi **low > high**.

+ Bắt buộc return: Cần có lệnh **return** ở mỗi lần gọi đệ quy để chuyển kết quả ngược trở lại từ các lệnh gọi sâu nhất.

2. Code C++:

```
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
using namespace std;         // Tiền tố std
int binary_search_recursive(vector<int>& arr, int target, int low, int high)
{
    // Thuc hien tim kiem Nhi phan bang phuong phap De quy.
    // 1. Truong hop co so (Base Case): Khong tim thay
    // Khi ranh gioi duoi vuot qua ranh gioi tren, pham vi tim kiem rong
    if (low > high)
    {
        return -1;
    }
    // Tinh chi muc giua
    int mid = (low + high) / 2;
    // 2. Truong hop tim thay
    if (arr[mid] == target)
    {
        return mid;
    }
    // 3. Target nam o nua trai (low den mid - 1)
    // Neu phan tu giua lon hon target, ta biet target phai nam ben trai (neu co)
    else if (arr[mid] > target)
    {
        // Goi de quy voi ranh gioi moi: high = mid - 1
        return binary_search_recursive(arr, target, low, mid - 1);
    }
    // 4. Target nam o nua phai (mid + 1 den high)
    // Neu phan tu giua nho hon target, ta biet target phai nam ben phai (neu co)
    else    // arr[mid] < target
    {
        // Goi de quy voi ranh gioi moi: low = mid + 1
        return binary_search_recursive(arr, target, mid + 1, high);
    }
}
int main()
{
    // ----------------- CHAY THU -----------------
    vector<int> arr = {2, 5, 8, 12, 16, 23, 38, 56, 72, 91};
    int target1 = 72;
    int target2 = 100;
    int n = arr.size();
    cout << "Du lieu da sap xep: ";
    for (int x : arr) cout << x << " ";
    cout << endl;
```

```
    cout << string(40, '-') << endl;
    // Chay thu 1: Tim thay
    int index1 = binary_search_recursive(arr, target1, 0, n - 1);
    cout << "Tim kiem " << target1 << ": Index = " << index1 << endl;
    // Chay thu 2: Khong tim thay
    int index2 = binary_search_recursive(arr, target2, 0, n - 1);
    cout << "Tim kiem " << target2 << ": Index = " << index2 << endl;
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
Du lieu da sap xep: 2 5 8 12 16 23 38 56 72 91
----------------------------------------
Tim kiem 72: Index = 8
Tim kiem 100: Index = -1
```

- Giải thích Code C++:

Tham số: Hàm đệ quy bắt buộc phải nhận **low** và **high** để xác định phạm vi làm việc của nó trong mỗi lần gọi. Cần truyền **const vector<int>& arr** để tránh sao chép mảng trong mỗi lần gọi.

VI. Tổng kết & Bài tập Về nhà (15 phút)

1. So sánh Dạng Lặp và Đệ quy:

Tiêu chí	Dạng Lặp (Iterative)	Dạng Đệ quy (Recursive)
Độ phức tạp	O(log N)	O(log N)
Bộ nhớ	O(1) (Tốt hơn)	O(log N) (Tốn bộ nhớ Stack Call)
Tốc độ	Nhanh hơn một chút (Không có overhead của hàm gọi)	Chậm hơn (do overhead của hàm gọi)
Độ rõ ràng	Phù hợp cho cài đặt sản xuất	Phù hợp cho việc học và hiểu nguyên lý

2. Bài tập Về nhà:

2.1. Bài tập 1 (Tìm kiếm Lần đầu/cuối): Sửa đổi **Binary Search** để tìm **lần xuất hiện đầu tiên** của một phần tử trùng lặp (**target**) trong mảng đã sắp xếp (ví dụ: tìm kiếm 5 trong [1, 5, 5, 5, 8]).

2.1.1. Code Python:

```python
def binary_search_first_occurrence(arr, target):
    # Sử dụng Binary Search để tìm chỉ mục của lần xuất hiện đầu tiên của target
    # trong mảng đã sắp xếp có chứa các phần tử trùng lặp.
    # Trả về: Chỉ mục đầu tiên tìm thấy, hoặc -1 nếu không tìm thấy.
    low = 0
    high = len(arr) - 1
    result = -1  # Lưu chỉ mục của lần tìm thấy gần nhất
    while low <= high:
        mid = (low + high) // 2
        if arr[mid] == target:
            # 1. Tìm thấy: Lưu kết quả tiềm năng
            result = mid
            # 2. Quan trọng: Tìm kiếm tiếp tục ở nửa trái
            # Bỏ qua nửa phải vì lần xuất hiện đầu tiên phải nằm bên trái của mid
            high = mid - 1
        elif arr[mid] < target:
            # Target lớn hơn, dịch chuyển ranh giới dưới sang phải
            low = mid + 1
        else: # arr[mid] > target
```

```
            # Target nhỏ hơn, dịch chuyển ranh giới trên sang trái
            high = mid - 1
    return result
# ----------------- CHẠY THỬ -----------------
arr = [1, 5, 5, 5, 8, 10, 10, 15, 20]
target1 = 5
target2 = 10
target3 = 1
print(f"Du lieu: {arr}")
print("-" * 40)
# Ví dụ 1: Tìm lần xuất hiện đầu tiên của 5
# Kết quả mong đợi: Index 1
index1 = binary_search_first_occurrence(arr, target1)
print(f"Lan dau xuat hien cua {target1}: Index = {index1}")
# Ví dụ 2: Tìm lần xuất hiện đầu tiên của 10
# Kết quả mong đợi: Index 5
index2 = binary_search_first_occurrence(arr, target2)
print(f"Lan dau xuat hien cua {target2}: Index = {index2}")
# Ví dụ 3: Phần tử không trùng lặp (trường hợp bình thường)
# Kết quả mong đợi: Index 0
index3 = binary_search_first_occurrence(arr, target3)
print(f"Lan dau xuat hien cua {target3}: Index = {index3}")
# Ví dụ 4: Không tìm thấy
index4 = binary_search_first_occurrence(arr, 99)
print(f"Lan dau xuat hien cua {99}: Index = {index4}")
```

=> Kết quả sau khi chạy chương trình:

```
Du lieu: [1, 5, 5, 5, 8, 10, 10, 15, 20]
----------------------------------------
Lan dau xuat hien cua 5: Index = 1
Lan dau xuat hien cua 10: Index = 5
Lan dau xuat hien cua 1: Index = 0
Lan dau xuat hien cua 99: Index = -1
```

2.1.2. Code C++:

```
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
using namespace std;         // Tiền tố std
int binary_search_first_occurrence(vector<int>& arr, int target)
{
    // Su dung Binary Search de tim chi muc cua lan xuat hien dau tien cua target
    // trong mang da sap xep co chua cac phan tu trung lap.
    // Tra ve: Chi muc dau tien tim thay, hoac -1 neu khong tim thay.
    int low = 0;
    int high = arr.size() - 1;
    int result = -1;  // Luu chi muc cua lan tim thay gan nhat
    while (low <= high)
    {
        int mid = (low + high) / 2;
        if (arr[mid] == target)
        {
            // 1. Tim thay: Luu ket qua tiem nang
            result = mid;
            // 2. Quan trong: Tim kiem tiep tuc o nua trai
            // Bo qua nua phai vi lan xuat hien dau tien phai nam ben trai cua mid
            high = mid - 1;
        }
        else if (arr[mid] < target)
        {
```

```
            // Target lon hon, dich chuyen ranh gioi duoi sang phai
            low = mid + 1;
        }
        else    // arr[mid] > target
        {
            // Target nho hon, dich chuyen ranh gioi tren sang trai
            high = mid - 1;
        }
    }
    return result;
}
int main()
{
    // ----------------- CHAY THU -----------------
    vector<int> arr = {1, 5, 5, 5, 8, 10, 10, 15, 20};
    int target1 = 5;
    int target2 = 10;
    int target3 = 1;
    cout << "Du lieu: ";
    for (int x : arr) cout << x << " ";
    cout << endl;
    cout << string(40, '-') << endl;
    // Vi du 1: Tim lan xuat hien dau tien cua 5
    // Ket qua mong doi: Index 1
    int index1 = binary_search_first_occurrence(arr, target1);
    cout << "Lan dau xuat hien cua " << target1 << ": Index = " << index1 << endl;
    // Vi du 2: Tim lan xuat hien dau tien cua 10
    // Ket qua mong doi: Index 5
    int index2 = binary_search_first_occurrence(arr, target2);
    cout << "Lan dau xuat hien cua " << target2 << ": Index = " << index2 << endl;
    // Vi du 3: Phan tu khong trung lap (truong hop binh thuong)
    // Ket qua mong doi: Index 0
    int index3 = binary_search_first_occurrence(arr, target3);
    cout << "Lan dau xuat hien cua " << target3 << ": Index = " << index3 << endl;
    // Vi du 4: Khong tim thay
    int index4 = binary_search_first_occurrence(arr, 99);
    cout << "Lan dau xuat hien cua 99: Index = " << index4 << endl;
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
Du lieu: 1 5 5 5 8 10 10 15 20
----------------------------------------
Lan dau xuat hien cua 5: Index = 1
Lan dau xuat hien cua 10: Index = 5
Lan dau xuat hien cua 1: Index = 0
Lan dau xuat hien cua 99: Index = -1
```

2.2. Bài tập 2 (Tìm kiếm trong Mảng xoay): Viết hàm tìm kiếm một phần tử trong một mảng đã sắp xếp bị xoay (Rotated Sorted Array), ví dụ: [8, 9, 1, 2, 3, 4, 5]. (Độ phức tạp mục tiêu: O(log N)).

2.2.1. Code Python:

```python
def binary_search_rotated(arr, target):
    # Thực hiện Binary Search trên mảng đã sắp xếp bị xoay (Rotated Sorted Array)
    # với độ phức tạp O(log N).
    # Trả về: Chỉ mục của target, hoặc -1 nếu không tìm thấy.
    low = 0
    high = len(arr) - 1
```

```
    while low <= high:
        mid = (low + high) // 2
        # Trường hợp cơ sở: Tìm thấy mục tiêu
        if arr[mid] == target:
            return mid
        # --- 1. Xác định xem NỬA TRÁI (low -> mid) có sắp xếp không ---
        if arr[low] <= arr[mid]:
            # Nửa trái đã được sắp xếp (ví dụ: [8, 9] trong [8, 9, 1, 2,
3])
            # Kiểm tra xem target có nằm trong nửa trái đã sắp xếp không
            if arr[low] <= target < arr[mid]:
                # Target nằm trong phạm vi sắp xếp: Bỏ qua nửa phải
                high = mid - 1
            else:
                # Target không nằm trong nửa trái: Chắc chắn phải nằm ở nửa phải bị
xoay
                low = mid + 1
        # --- 2. Nếu NỬA TRÁI không sắp xếp, thì NỬA PHẢI (mid -> high) phải sắp
xếp ---
        else:
            # Nửa phải đã được sắp xếp (ví dụ: [1, 2, 3] trong [9, 1, 2,
3])
            # Kiểm tra xem target có nằm trong nửa phải đã sắp xếp không
            if arr[mid] < target <= arr[high]:
                # Target nằm trong phạm vi sắp xếp: Bỏ qua nửa trái
                low = mid + 1
            else:
                # Target không nằm trong nửa phải: Chắc chắn phải nằm ở nửa trái bị
xoay
                high = mid - 1
    # Nếu vòng lặp kết thúc mà không tìm thấy
    return -1
# ----------------- CHẠY THỬ -----------------
arr_rotated = [8, 9, 1, 2, 3, 4, 5]
target1 = 3
target2 = 8
target3 = 10
print(f"Mang da sap xep bi xoay: {arr_rotated}")
print("-" * 40)
# Ví dụ 1: Tìm kiếm trong nửa phải đã sắp xếp
index1 = binary_search_rotated(arr_rotated, target1)
print(f"Tim kiem {target1}: Index = {index1}") # Kết quả: 4
# Ví dụ 2: Tìm kiếm trong nửa trái đã sắp xếp
index2 = binary_search_rotated(arr_rotated, target2)
print(f"Tim kiem {target2}: Index = {index2}") # Kết quả: 0
# Ví dụ 3: Không tìm thấy
index3 = binary_search_rotated(arr_rotated, target3)
print(f"Tim kiem {target3}: Index = {index3}") # Kết quả: -1
```

=> Kết quả sau khi chạy chương trình:

```
Mang da sap xep bi xoay: [8, 9, 1, 2, 3, 4, 5]
----------------------------------------
Tim kiem 3: Index = 4
Tim kiem 8: Index = 0
Tim kiem 10: Index = -1
```

2.2.2. Code C++:

```
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
using namespace std;         // Tiền tố std
int binary_search_rotated(vector<int>& arr, int target)
{
    // Thuc hien Binary Search tren mang da sap xep bi xoay (Rotated Sorted Array)
    // voi do phuc tap O(log N).
    // Tra ve: Chi muc cua target, hoac -1 neu khong tim thay.
    int low = 0;
    int high = arr.size() - 1;
    while (low <= high)
    {
        int mid = (low + high) / 2;
        // Truong hop co so: Tim thay muc tieu
        if (arr[mid] == target)
        {
            return mid;
        }
        // --- 1. Xac dinh xem NUA TRAI (low -> mid) co sap xep khong ---
        if (arr[low] <= arr[mid])
        {
            // Nua trai da duoc sap xep (vi du: [8, 9] trong [8, 9, 1, 2, 3])
            // Kiem tra xem target co nam trong nua trai da sap xep khong
            if (arr[low] <= target && target < arr[mid])
            {
                // Target nam trong pham vi sap xep: Bo qua nua phai
                high = mid - 1;
            }
            else
            {
                // Target khong nam trong nua trai: Chac chan phai nam o nua phai
bi xoay
                low = mid + 1;
            }
        }
        // --- 2. Neu NUA TRAI khong sap xep, thi NUA PHAI (mid -> high) phai sap
xep ---
        else
        {
            // Nua phai da duoc sap xep (vi du: [1, 2, 3] trong [9, 1, 2, 3])
            // Kiem tra xem target co nam trong nua phai da sap xep khong
            if (arr[mid] < target && target <= arr[high])
            {
                // Target nam trong pham vi sap xep: Bo qua nua trai
                low = mid + 1;
            }
            else
            {
                // Target khong nam trong nua phai: Chac chan phai nam o nua trai
bi xoay
                high = mid - 1;
            }
        }
    }
    // Neu vong lap ket thuc ma khong tim thay
    return -1;
```

```
}
int main()
{
    // ----------------- CHAY THU -----------------
    vector<int> arr_rotated = {8, 9, 1, 2, 3, 4, 5};
    int target1 = 3;
    int target2 = 8;
    int target3 = 10;
    cout << "Mang da sap xep bi xoay: ";
    for (int x : arr_rotated) cout << x << " ";
    cout << endl;
    cout << string(40, '-') << endl;
    // Vi du 1: Tim kiem trong nua phai da sap xep
    int index1 = binary_search_rotated(arr_rotated, target1);
    cout << "Tim kiem " << target1 << ": Index = " << index1 << endl;
    // Vi du 2: Tim kiem trong nua trai da sap xep
    int index2 = binary_search_rotated(arr_rotated, target2);
    cout << "Tim kiem " << target2 << ": Index = " << index2 << endl;
    // Vi du 3: Khong tim thay
    int index3 = binary_search_rotated(arr_rotated, target3);
    cout << "Tim kiem " << target3 << ": Index = " << index3 << endl;
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
Mang da sap xep bi xoay: 8 9 1 2 3 4 5
----------------------------------------
Tim kiem 3: Index = 4
Tim kiem 8: Index = 0
Tim kiem 10: Index = -1
```

2.3. Bài tập 3 (Floor và Ceiling): Sửa đổi Binary Search để tìm **Floor** (**phần tử lớn nhất ≤ target**) và **Ceiling** (**phần tử nhỏ nhất ≥ target**) của một giá trị trong mảng.

2.3.1. Code Python:

```
def find_floor_ceiling(arr, target):
    # Sử dụng Binary Search để tìm Floor (lớn nhất <= target) và
    # Ceiling (nhỏ nhất >= target) của một giá trị trong mảng đã sắp xếp.
    # Trả về: Tuple (Floor, Ceiling). Trả về None nếu không tìm thấy.
    low = 0
    high = len(arr) - 1
    # Khởi tạo giá trị Floor và Ceiling:
    # None biểu thị rằng chưa tìm thấy giá trị hợp lệ
    floor = None
    ceiling = None
    while low <= high:
        mid = (low + high) // 2
        mid_val = arr[mid]
        # --- Trường hợp 1: arr[mid] == target (Tìm thấy) ---
        if mid_val == target:
            return mid_val, mid_val # Floor và Ceiling chính là target
        # --- Trường hợp 2: arr[mid] < target (Target nằm ở nửa phải) ---
        elif mid_val < target:
            # mid_val là Floor tiềm năng (vì mid_val < target)
            floor = mid_val
            # Tìm kiếm ở nửa phải để tìm Floor lớn hơn
            low = mid + 1
        # --- Trường hợp 3: arr[mid] > target (Target nằm ở nửa trái) ---
        else: # mid_val > target
```

```
            # mid_val là Ceiling tiềm năng (vì mid_val > target)
            ceiling = mid_val
            # Tìm kiếm ở nửa trái để tìm Ceiling nhỏ hơn
            high = mid - 1
    # Trả về kết quả tìm được sau khi vòng lặp kết thúc
    return floor, ceiling
# ----------------- CHẠY THỬ -----------------
arr = [5, 10, 15, 20, 25, 30]
print(f"Du lieu: {arr}\n" + "-" * 40)
# Ví dụ 1: Target nằm trong mảng
target1 = 15
f1, c1 = find_floor_ceiling(arr, target1)
print(f"Target: {target1} -> Floor: {f1}, Ceiling: {c1}")
# Ví dụ 2: Target không có trong mảng (Giữa hai giá trị 15 và 20)
target2 = 18
f2, c2 = find_floor_ceiling(arr, target2)
print(f"Target: {target2} -> Floor: {f2}, Ceiling: {c2}")
# Ví dụ 3: Target nhỏ hơn tất cả (Không có Floor)
target3 = 2
f3, c3 = find_floor_ceiling(arr, target3)
print(f"Target: {target3} -> Floor: {f3}, Ceiling: {c3}")
# Ví dụ 4: Target lớn hơn tất cả (Không có Ceiling)
target4 = 35
f4, c4 = find_floor_ceiling(arr, target4)
print(f"Target: {target4} -> Floor: {f4}, Ceiling: {c4}")
```

=> Kết quả sau khi chạy chương trình:

```
Du lieu: [5, 10, 15, 20, 25, 30]
----------------------------------------
Target: 15 -> Floor: 15, Ceiling: 15
Target: 18 -> Floor: 15, Ceiling: 20
Target: 2 -> Floor: None, Ceiling: 5
Target: 35 -> Floor: 30, Ceiling: None
```

2.3.2. Code C++:

```
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
using namespace std;         // Tiền tố std
pair<int, int> find_floor_ceiling(vector<int>& arr, int target)
{
    // Su dung Binary Search de tim Floor (lon nhat <= target) va
    // Ceiling (nho nhat >= target) cua mot gia tri trong mang da sap xep.
    // Tra ve: Pair (Floor, Ceiling). Tra ve {INT_MIN, INT_MIN} neu khong tim thay.
    int low = 0;
    int high = arr.size() - 1;
    // Khoi tao gia tri Floor va Ceiling:
    // INT_MIN bieu thi rang chua tim thay gia tri hop le
    int floor = INT_MIN;
    int ceiling = INT_MIN;
    while (low <= high)
    {
        int mid = (low + high) / 2;
        int mid_val = arr[mid];
        // --- Truong hop 1: arr[mid] == target (Tim thay) ---
        if (mid_val == target)
        {
            return {mid_val, mid_val}; // Floor va Ceiling chinh la target
        }
        // --- Truong hop 2: arr[mid] < target (Target nam o nua phai) ---
```

```
        else if (mid_val < target)
        {
            // mid_val la Floor tiem nang (vi mid_val < target)
            floor = mid_val;
            // Tim kiem o nua phai de tim Floor lon hon
            low = mid + 1;
        }
        // --- Truong hop 3: arr[mid] > target (Target nam o nua trai) ---
        else    // mid_val > target
        {
            // mid_val la Ceiling tiem nang (vi mid_val > target)
            ceiling = mid_val;
            // Tim kiem o nua trai de tim Ceiling nho hon
            high = mid - 1;
        }
    }
    // Tra ve ket qua tim duoc sau khi vong lap ket thuc
    return {floor, ceiling};
}
int main()
{
    // ----------------- CHAY THU -----------------
    vector<int> arr = {5, 10, 15, 20, 25, 30};
    cout << "Du lieu: ";
    for (int x : arr) cout << x << " ";
    cout << "\n" << string(40, '-') << endl;
    // Vi du 1: Target nam trong mang
    int target1 = 15;
    auto [f1, c1] = find_floor_ceiling(arr, target1);
    cout << "Target: " << target1 << " -> Floor: " << f1 << ", Ceiling: " << c1 <<
endl;
    // Vi du 2: Target khong co trong mang (Giua hai gia tri 15 va 20)
    int target2 = 18;
    auto [f2, c2] = find_floor_ceiling(arr, target2);
    cout << "Target: " << target2 << " -> Floor: " << f2 << ", Ceiling: " << c2 <<
endl;
    // Vi du 3: Target nho hon tat ca (Khong co Floor)
    int target3 = 2;
    auto [f3, c3] = find_floor_ceiling(arr, target3);
    cout << "Target: " << target3 << " -> Floor: " << f3 << ", Ceiling: " << c3 <<
endl;
    // Vi du 4: Target lon hon tat ca (Khong co Ceiling)
    int target4 = 35;
    auto [f4, c4] = find_floor_ceiling(arr, target4);
    cout << "Target: " << target4 << " -> Floor: " << f4 << ", Ceiling: " << c4 <<
endl;
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
Du lieu: 5 10 15 20 25 30
----------------------------------------
Target: 15 -> Floor: 15, Ceiling: 15
Target: 18 -> Floor: 15, Ceiling: 20
Target: 2 -> Floor: -2147483648, Ceiling: 5
Target: 35 -> Floor: 30, Ceiling: -2147483648
```

- Giải thích Code C++:

+ Trong Python: None => Trong C++ phải thay bằng giá trị đặc biệt. Cách này dùng INT_MIN để biểu diễn không tồn tại.

+ Nếu floor = INT_MIN → Không có Floor

+ Nếu ceiling = INT_MIN → Không có Ceiling

CHỦ ĐỀ 8. CẤU TRÚC DỮ LIỆU

BÀI 22. NGĂN XẾP (STACK)

I. Mục Tiêu:

- **Kiến thức:** HS hiểu rõ **Stack** là cấu trúc dữ liệu tuyến tính hoạt động theo nguyên tắc **LIFO (Last-In, First-Out)**. Nắm vững các thao tác cơ bản: **Push, Pop, Peek, isEmpty**.

- **Kỹ năng:** HS cài đặt được **Stack** sử dụng Mảng/Danh sách động (List/Vector) trong Python và C++. HS áp dụng **Stack** để giải quyết các bài toán đơn giản (ví dụ: Đảo ngược chuỗi).

- **Tư duy:** HS so sánh được **Stack** với **Queue** (Nguyên tắc FIFO) và nhận biết các trường hợp sử dụng Stack trong thực tế (Ví dụ: Chức năng Undo/Redo, quản lý hàm gọi).

II. Phân Bổ Thời Gian (90 Phút):

Hoạt Động	Nội Dung	Thời gian (Phút)
Ôn tập	Ôn tập QuickSort và Binary Search	10
Lý thuyết 1	Stack: Định nghĩa, Nguyên tắc LIFO	15
Thực hành 1	Bài 1. Cài đặt Stack (Sử dụng List/Vector)	30
Thực hành 2	Bài 2. Ứng dụng Stack: Đảo ngược Chuỗi	25
Tổng kết	So sánh, Bài tập về nhà	10
Tổng cộng		**90**

III. Lý Thuyết 1: Stack - Nguyên tắc LIFO (15 phút)

1. Định nghĩa và Nguyên tắc:

- **Stack (Ngăn xếp)** là một cấu trúc dữ liệu trừu tượng (Abstract Data Type - ADT) hoạt động theo nguyên tắc: **LIFO (Last-In, First-Out)**

- Nguyên tắc này giống như một chồng đĩa hoặc chồng sách: Phần tử được đặt vào **cuối cùng** sẽ là phần tử được lấy ra **đầu tiên**.

- Các thao tác được thực hiện tại một đầu duy nhất gọi là **Top (Đỉnh)** của Stack.

2. Các Thao tác Cơ bản:

Thao tác	Mô tả	Độ phức tạp
Push	Thêm một phần tử mới vào **đỉnh** (Top) của Stack.	O(1)
Pop	Loại bỏ và trả về phần tử ở **đỉnh** (Top) của Stack.	O(1)

Peek	Chỉ xem và trả về phần tử ở **đỉnh** (Top) mà không loại bỏ nó.	O(1)
isEmpty	Kiểm tra xem Stack có rỗng hay không.	O(1)
Size	Trả về số lượng phần tử hiện tại trong Stack.	O(1)

IV. Thực Hành 1: Cài đặt Stack (30 phút)

Ta sẽ cài đặt Stack sử dụng **Danh sách/Mảng động** (Python list, C++ vector) vì chúng cho phép thao tác Push và Pop ở cuối danh sách với độ phức tạp O(1) trung bình.

1. Bài 1: Cài đặt Stack Class

1.1. Code Python:

```python
class Stack:
    def __init__(self):
        # Khởi tạo: Sử dụng list Python làm bộ lưu trữ (phần tử cuối là đỉnh Stack)
        self.items = []
    def is_empty(self):
        # Kiểm tra xem list có rỗng không
        return not self.items
    def push(self, item):
        # Thêm phần tử vào cuối list (Đỉnh Stack). O(1) trung bình
        self.items.append(item)
    def pop(self):
        # Lấy và xóa phần tử cuối cùng (Đỉnh Stack). O(1) trung bình
        if not self.is_empty():
            return self.items.pop()
        else:
            # Nên sử dụng IndexError thay vì Exception chung
            raise IndexError("Loi: Stack rong, khong the Pop")
    def peek(self):
        # Xem phần tử cuối cùng (Đỉnh Stack) mà không xóa. O(1)
        if not self.is_empty():
            return self.items[-1]
        else:
            raise IndexError("Loi: Stack rong, khong the Peek")
# Chạy thử
stack = Stack()
stack.push(10)
stack.push(20)
print(f"Cac phan tu trong Stack: {stack.items}")
print("Peek (Dinh hien tai):", stack.peek())
# Sẽ lấy 20 ra
print("Pop (Phan tu lay ra):", stack.pop())
# Stack còn lại [10]
print(f"Stack sau Pop: {stack.items}")
print("IsEmpty (Kiem tra rong):", stack.is_empty())
stack.pop() # Lấy 10 ra
print("IsEmpty (Sau khi Pop het):", stack.is_empty())
```

=> Kết quả sau khi chạy chương trình:

```
Cac phan tu trong Stack: [10, 20]
Peek (Dinh hien tai): 20
Pop (Phan tu lay ra): 20
Stack sau Pop: [10]
IsEmpty (Kiem tra rong): False
IsEmpty (Sau khi Pop het): True
```

- Giải thích Code Python:

Sử dụng **list.append()** và **list.pop()**: Các thao tác này trên list Python (thao tác ở cuối) có độ phức tạp trung bình O(1), lý tưởng cho Stack.

1.2. Code C++:

```
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
using namespace std;         // Tiền tố std
class Stack
{
private:
    // Khoi tao: Su dung vector C++ lam bo luu tru (phan tu cuoi la dinh Stack)
    vector<int> items;
public:
    Stack()
    {
        // Ham khoi tao mac dinh
    }
    bool is_empty()
    {
        // Kiem tra xem vector co rong khong
        return items.empty();
    }
    void push(int item)
    {
        // Them phan tu vao cuoi vector (Dinh Stack). O(1) trung binh
        items.push_back(item);
    }
    int pop()
    {
        // Lay va xoa phan tu cuoi cung (Dinh Stack). O(1) trung binh
        if (!is_empty())
        {
            int top_item = items.back();
            items.pop_back();
            return top_item;
        }
        else
        {
            // Nem loi khi Stack rong
            throw out_of_range("Loi: Stack rong, khong the Pop");
        }
    }
    int peek()
    {
        // Xem phan tu cuoi cung (Dinh Stack) ma khong xoa. O(1)
        if (!is_empty())
        {
            return items.back();
        }
        else
        {
            throw out_of_range("Loi: Stack rong, khong the Peek");
        }
    }
    // Ham phu de in cac phan tu trong Stack (phuc vu chay thu)
    void print_stack()
```

```
    {
        cout << "Cac phan tu trong Stack: ";
        for (int x : items) cout << x << " ";
        cout << endl;
    }
};
int main()
{
    // Chay thu
    Stack stack;
    stack.push(10);
    stack.push(20);
    stack.print_stack();
    cout << "Peek (Dinh hien tai): " << stack.peek() << endl;
    // Se lay 20 ra
    cout << "Pop (Phan tu lay ra): " << stack.pop() << endl;
    // Stack con lai [10]
    stack.print_stack();
    cout << "IsEmpty (Kiem tra rong): " << stack.is_empty() << endl;
    stack.pop(); // Lay 10 ra
    cout << "IsEmpty (Sau khi Pop het): " << stack.is_empty() << endl;
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
Cac phan tu trong Stack: 10 20
Peek (Dinh hien tai): 20
Pop (Phan tu lay ra): 20
Cac phan tu trong Stack: 10
IsEmpty (Kiem tra rong): 0
IsEmpty (Sau khi Pop het): 1
```

- Giải thích Code C++:

+ **Sử dụng vector:** Tương tự Python**, vector::push_back()** và **vector::pop_back()** cho hiệu suất **O(1).**

+ **back():** Được dùng để lấy giá trị đỉnh **(Peek** và **Pop** trước khi xóa**)**.

V. Thực Hành 2: Ứng dụng Stack (25 phút)

- Một ứng dụng kinh điển của Stack là **Đảo ngược một chuỗi** (hoặc một mảng).

- **Nguyên lý:** Đẩy lần lượt từng ký tự (hoặc phần tử) của chuỗi/mảng vào Stack. Khi lấy ra (Pop), do nguyên tắc LIFO, các ký tự sẽ được lấy ra theo thứ tự ngược lại.

1. Bài 2: Đảo ngược Chuỗi

1.1. Code Python:

```
class Stack:
    def __init__(self):
        # Khởi tạo: Sử dụng list Python làm bộ lưu trữ (phần tử cuối là đỉnh Stack)
        self.items = []
    def is_empty(self):
        # Kiểm tra xem list có rỗng không
        return not self.items
    def push(self, item):
        # Thêm phần tử vào cuối list (Đỉnh Stack). O(1) trung bình
        self.items.append(item)
    def pop(self):
        # Lấy và xóa phần tử cuối cùng (Đỉnh Stack). O(1) trung bình
        if not self.is_empty():
            return self.items.pop()
```

```
        else:
            raise IndexError("Loi: Stack rong, khong the Pop")
    def peek(self):
        # Xem phần tử cuối cùng (Đỉnh Stack) mà không xóa. O(1)
        if not self.is_empty():
            return self.items[-1]
        else:
            raise IndexError("Loi: Stack rong, khong the Peek")
def reverse_string_with_stack(s):
    stack = Stack() # Sử dụng lớp Stack đã định nghĩa
    # KHỞI TẠO ĐÚNG: Biến phải nằm ngoài vòng lặp
    reversed_s = ""
    # 1. Push: Đẩy từng ký tự vào Stack (LIFO)
    # Chuỗi gốc "HELLO" -> Stack: H, E, L, L, O (O ở trên cùng)
    for char in s:
        stack.push(char)
    # 2. Pop: Lấy từng ký tự ra khỏi Stack (theo thứ tự LIFO)
    # Lấy ra: O, L, L, E, H
    while not stack.is_empty():
        reversed_s += stack.pop()
    return reversed_s
# Chạy thử
original = "HELLO"
reversed_str = reverse_string_with_stack(original)
print(f"Chuoi goc: {original}")
print(f"Chuoi dao nguoc: {reversed_str}") # Kết quả: OLLEH (Đúng)
```

=> Kết quả sau khi chạy chương trình:

```
Chuoi goc: HELLO
Chuoi dao nguoc: OLLEH
```

- Giải thích Code Python:

Tận dụng LIFO: Chuỗi gốc "HELLO" được Push theo thứ tự H, E, L, L, O. Khi Pop, thứ tự là O, L, L, E, H.

1.2. Code C++:

```
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
using namespace std;         // Tiền tố std
class Stack
{
private:
    // Khoi tao: Su dung vector C++ lam bo luu tru (phan tu cuoi la dinh Stack)
    vector<char> items;
public:
    Stack()
    {
        // Ham khoi tao mac dinh
    }
    bool is_empty()
    {
        // Kiem tra xem vector co rong khong
        return items.empty();
    }
    void push(char item)
    {
        // Them phan tu vao cuoi vector (Dinh Stack). O(1) trung binh
        items.push_back(item);
    }
```

```
    char pop()
    {
        // Lay va xoa phan tu cuoi cung (Dinh Stack). O(1) trung binh
        if (!is_empty())
        {
            char top_item = items.back();
            items.pop_back();
            return top_item;
        }
        else
        {
            throw out_of_range("Loi: Stack rong, khong the Pop");
        }
    }
    char peek()
    {
        // Xem phan tu cuoi cung (Dinh Stack) ma khong xoa. O(1)
        if (!is_empty())
        {
            return items.back();
        }
        else
        {
            throw out_of_range("Loi: Stack rong, khong the Peek");
        }
    }
};
string reverse_string_with_stack(string s)
{
    Stack stack; // Su dung lop Stack da dinh nghia
    // KHOI TAO DUNG: Bien phai nam ngoai vong lap
    string reversed_s = "";
    // 1. Push: Day tung ky tu vao Stack (LIFO)
    // Chuoi goc "HELLO" -> Stack: H, E, L, L, O (O o tren cung)
    for (char c : s)
    {
        stack.push(c);
    }
    // 2. Pop: Lay tung ky tu ra khoi Stack (theo thu tu LIFO)
    // Lay ra: O, L, L, E, H
    while (!stack.is_empty())
    {
        reversed_s += stack.pop();
    }
    return reversed_s;
}
int main()
{
    // Chay thu
    string original = "HELLO";
    string reversed_str = reverse_string_with_stack(original);
    cout << "Chuoi goc: " << original << endl;
    cout << "Chuoi dao nguoc: " << reversed_str << endl; // Ket qua: OLLEH (Dung)
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
Chuoi goc: HELLO
Chuoi dao nguoc: OLLEH
```

- Giải thích Code C++:

stack: Thư viện chuẩn C++ cung cấp **stack**, đơn giản hóa việc sử dụng. Các hàm tương ứng: push(), pop() (chỉ xóa), top() (chỉ xem), empty().

VI. Tổng kết & Bài tập Về nhà (10 phút)

1. Tóm tắt:

Stack là cấu trúc dữ liệu cơ bản, được cài đặt hiệu quả nhất với O(1) cho các thao tác chính. Nguyên tắc LIFO khiến nó trở nên lý tưởng cho các tình huống đòi hỏi việc hoàn tác, kiểm tra thứ tự đóng/mở (dấu ngoặc), hoặc quản lý các tác vụ tạm thời.

2. Bài tập Về nhà:

2.1. Bài tập 1 (Kiểm tra Ngoặc): Viết chương trình sử dụng Stack để kiểm tra xem một biểu thức toán học có đóng mở ngoặc **cân bằng** hay không (ví dụ: { [( )] } là cân bằng, {[)} là không cân bằng).

2.1.1. Code Python:

```python
class Stack:
    # Cấu trúc Stack đơn giản sử dụng list.
    def __init__(self):
        self.items = []
    def push(self, item):
        self.items.append(item)
    def pop(self):
        if not self.is_empty():
            return self.items.pop()
        return None # Trả về None thay vì lỗi khi Stack rỗng
    def is_empty(self):
        return not self.items
def check_parentheses_balance(expression):
    # Kiểm tra tính cân bằng của các dấu ngoặc trong biểu thức.
    # Tham số:
    #   expression (str): Chuỗi chứa biểu thức toán học.
    # Trả về:
    #   bool: True nếu cân bằng, False nếu không.
    stack = Stack()
    # Định nghĩa các cặp ngoặc hợp lệ
    opening_brackets = "([{"
    closing_brackets = ")]}"
    # Map các dấu đóng với dấu mở tương ứng
    # Ví dụ: closing_map[')'] = '('
    closing_map = {
        ')': '(',
        ']': '[',
        '}': '{'
    }
    # Duyệt qua từng ký tự trong biểu thức
    for char in expression:
        # 1. Nếu là dấu mở, đẩy vào Stack
        if char in opening_brackets:
            stack.push(char)
        # 2. Nếu là dấu đóng
        elif char in closing_brackets:
            # 2a. Trường hợp 1: Stack rỗng
            # Gặp dấu đóng mà không có dấu mở tương ứng trong Stack
            if stack.is_empty():
```

```
                return False
            # 2b. Trường hợp 2: Kiểm tra khớp
            # Lấy dấu mở ở đỉnh Stack ra
            top_bracket = stack.pop()
            # So sánh dấu mở vừa lấy ra với dấu mở kỳ vọng của dấu đóng hiện tại
            if top_bracket != closing_map[char]:
                return False
    # 3. Kết thúc: Sau khi duyệt hết biểu thức
    # Nếu Stack rỗng, mọi dấu mở đều đã được đóng khớp.
    return stack.is_empty()
# ----------------- CHẠY THỬ -----------------
# Ví dụ cân bằng
exp1 = "{[()]} + (a * b)"
exp2 = "([{}])"
# Ví dụ không cân bằng
exp3 = "{[)}"  # Ngoặc không khớp
exp4 = "((abc)" # Thiếu ngoặc đóng
exp5 = ")} + 1" # Thừa ngoặc đóng
print(f"'{exp1}' -> Can bang: {check_parentheses_balance(exp1)}")
print(f"'{exp2}' -> Can bang: {check_parentheses_balance(exp2)}")
print("-" * 30)
print(f"'{exp3}' -> Can bang: {check_parentheses_balance(exp3)}")
print(f"'{exp4}' -> Can bang: {check_parentheses_balance(exp4)}")
print(f"'{exp5}' -> Can bang: {check_parentheses_balance(exp5)}")
```

=> Kết quả sau khi chạy chương trình:

```
'{[()]} + (a * b)' -> Can bang: True
'([{}])' -> Can bang: True
------------------------------
'{[)}' -> Can bang: False
'((abc)' -> Can bang: False
')} + 1' -> Can bang: False
```

2.1.2. Code C++:

```
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
using namespace std;         // Tiền tố std
class Stack
{
    // Cau truc Stack don gian su dung vector.
private:
    vector<char> items;
public:
    Stack()
    {
        // Ham khoi tao
    }
    void push(char item)
    {
        items.push_back(item);
    }
    char pop()
    {
        if (!is_empty())
        {
            char top_item = items.back();
            items.pop_back();
            return top_item;
        }
```

```
        return '\0'; // Tra ve ky tu rong thay vi loi khi Stack rong
    }
    bool is_empty()
    {
        return items.empty();
    }
};
bool check_parentheses_balance(string expression)
{
    // Kiem tra tinh can bang cua cac dau ngoac trong bieu thuc.
    // Tham so:
    //   expression (string): Chuoi chua bieu thuc toan hoc.
    // Tra ve:
    //   bool: True neu can bang, False neu khong.
    Stack stack;
    // Dinh nghia cac cap ngoac hop le
    string opening_brackets = "([{";
    string closing_brackets = ")]}";
    // Map cac dau dong voi dau mo tuong ung
    // Vi du: closing_map[')'] = '('
    unordered_map<char, char> closing_map = {
        {')', '('},
        {']', '['},
        {'}', '{'}
    };
    // Duyet qua tung ky tu trong bieu thuc
    for (char ch : expression)
    {
        // 1. Neu la dau mo, day vao Stack
        if (opening_brackets.find(ch) != string::npos)
        {
            stack.push(ch);
        }
        // 2. Neu la dau dong
        else if (closing_brackets.find(ch) != string::npos)
        {
            // 2a. Truong hop 1: Stack rong
            // Gap dau dong ma khong co dau mo tuong ung trong Stack
            if (stack.is_empty())
            {
                return false;
            }
            // 2b. Truong hop 2: Kiem tra khop
            // Lay dau mo o dinh Stack ra
            char top_bracket = stack.pop();
            // So sanh dau mo vua lay ra voi dau mo ky vong
            if (top_bracket != closing_map[ch])
            {
                return false;
            }
        }
    }
    // 3. Ket thuc: Sau khi duyet het bieu thuc
    // Neu Stack rong, moi dau mo deu da duoc dong khop
    return stack.is_empty();
}
int main()
{
```

```
    // ---------------- CHAY THU ----------------
    // Vi du can bang
    string exp1 = "{[()]} + (a * b)";
    string exp2 = "([{}])";
    // Vi du khong can bang
    string exp3 = "{[)}";   // Ngoac khong khop
    string exp4 = "((abc)"; // Thieu ngoac dong
    string exp5 = ")} + 1"; // Thua ngoac dong
    cout << "'" << exp1 << "' -> Can bang: "
         << check_parentheses_balance(exp1) << endl;
    cout << "'" << exp2 << "' -> Can bang: "
         << check_parentheses_balance(exp2) << endl;
    cout << string(30, '-') << endl;
    cout << "'" << exp3 << "' -> Can bang: "
         << check_parentheses_balance(exp3) << endl;
    cout << "'" << exp4 << "' -> Can bang: "
         << check_parentheses_balance(exp4) << endl;
    cout << "'" << exp5 << "' -> Can bang: "
         << check_parentheses_balance(exp5) << endl;
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
'{[()]} + (a * b)' -> Can bang: True
'([{}])' -> Can bang: True
------------------------------
'{[)}' -> Can bang: False
'((abc)' -> Can bang: False
')} + 1' -> Can bang: False
```

2.2. Bài tập 2 (Chuyển đổi Hậu tố): Sử dụng Stack để chuyển đổi một biểu thức Infix (ví dụ: A + B * C) sang biểu thức Postfix (Hậu tố).

2.2.1. Code Python:

```
class Stack:
    # Cấu trúc Stack đơn giản sử dụng list.
    def __init__(self):
        self.items = []
    def push(self, item):
        self.items.append(item)
    def pop(self):
        if not self.is_empty():
            return self.items.pop()
        return None
    def peek(self):
        if not self.is_empty():
            return self.items[-1]
        return None
    def is_empty(self):
        return not self.items
def get_precedence(operator):
    # Xác định độ ưu tiên của toán tử.
    if operator == '^':
        return 3
    elif operator == '*' or operator == '/':
        return 2
    elif operator == '+' or operator == '-':
        return 1
    else:
```

```
        return 0
def infix_to_postfix(infix_expression):
    # Chuyển đổi biểu thức Infix sang Postfix bằng Stack.
    stack = Stack()
    postfix_output = []
    # Định nghĩa các toán tử
    operators = "+-*/^"
    # Duyệt qua từng ký tự trong biểu thức Infix
    for token in infix_expression:
        # Bỏ qua khoảng trắng
        if token == ' ':
            continue
        # 1. Toán hạng (Giả sử là chữ cái)
        elif 'A' <= token <= 'Z' or 'a' <= token <= 'z':
            postfix_output.append(token)
        # 2. Dấu mở ngoặc
        elif token == '(':
            stack.push(token)
        # 3. Dấu đóng ngoặc
        elif token == ')':
            # Pop toán tử cho đến khi gặp dấu '('
            while not stack.is_empty() and stack.peek() != '(':
                postfix_output.append(stack.pop())
            # Pop dấu '(' ra khỏi Stack (nếu có)
            if not stack.is_empty() and stack.peek() == '(':
                stack.pop()
        # 4. Toán tử
        elif token in operators:
            # Pop toán tử từ Stack ra Output nếu:
            # - Stack không rỗng
            # - Đỉnh Stack không phải là '('
            # - Độ ưu tiên của đỉnh Stack >= Độ ưu tiên của toán tử hiện tại
            while (not stack.is_empty() and stack.peek() != '(' and
                   get_precedence(stack.peek()) >= get_precedence(token)):
                postfix_output.append(stack.pop())
            # Đẩy toán tử hiện tại vào Stack
            stack.push(token)
    # 5. Kết thúc: Pop tất cả các toán tử còn lại trong Stack
    while not stack.is_empty():
        # Nếu gặp '(' ở đây, biểu thức Infix bị sai ngoặc
        if stack.peek() == '(':
            return "Bieu thuc Infix khong hop le (Du dau '(')"
        postfix_output.append(stack.pop())
    return "".join(postfix_output)
# ----------------- CHẠY THỬ -----------------
# Biểu thức ví dụ: A + B * C
infix1 = "A+B*C"
# Kết quả mong đợi: ABC*+
print(f"Infix: {infix1} -> Postfix: {infix_to_postfix(infix1)}")
# Biểu thức phức tạp hơn: (A + B) * (C - D)
infix2 = "(A+B)*(C-D)"
# Kết quả mong đợi: AB+CD-*
print(f"Infix: {infix2} -> Postfix: {infix_to_postfix(infix2)}")
# Biểu thức có lũy thừa và ngoặc: A * (B + C / D) - E
infix3 = "A*(B+C/D)-E"
# Kết quả mong đợi: ABCD/+*E-
print(f"Infix: {infix3} -> Postfix: {infix_to_postfix(infix3)}")
# Trường hợp xấu (Dư ngoặc mở):
```

```
infix4 = "((A+B)*C"
print(f"Infix: {infix4} -> Postfix: {infix_to_postfix(infix4)}")
```

=> Kết quả sau khi chạy chương trình:

```
Infix: A+B*C -> Postfix: ABC*+
Infix: (A+B)*(C-D) -> Postfix: AB+CD-*
Infix: A*(B+C/D)-E -> Postfix: ABCD/+*E-
Infix: ((A+B)*C -> Postfix: Bieu thuc Infix khong hop le (Du dau '(')
```

2.2.2. Code C++:

```
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
using namespace std;         // Tiền tố std
class Stack
{
    // Cau truc Stack don gian su dung vector.
private:
    vector<char> items;
public:
    Stack()
    {
        // Ham khoi tao
    }
    void push(char item)
    {
        items.push_back(item);
    }
    char pop()
    {
        if (!is_empty())
        {
            char top_item = items.back();
            items.pop_back();
            return top_item;
        }
        return '\0';
    }
    char peek()
    {
        if (!is_empty())
        {
            return items.back();
        }
        return '\0';
    }
    bool is_empty()
    {
        return items.empty();
    }
};
int get_precedence(char oper)
{
    // Xac dinh do uu tien cua toan tu.
    if (oper == '^')
    {
        return 3;
    } else if (oper == '*' || oper == '/')
    {
        return 2;
```

```
    } else if (oper == '+' || oper == '-')
    {
        return 1;
    } else
    {
        return 0;
    }
}
string infix_to_postfix(string infix_expression)
{
    // Chuyen doi bieu thuc Infix sang Postfix bang Stack.
    Stack stack;
    vector<char> postfix_output;
    // Dinh nghia cac toan tu
    string operators = "+-*/^";
    // Duyet qua tung ky tu trong bieu thuc Infix
    for (char token : infix_expression)
    {
        // Bo qua khoang trang
        if (token == ' ')
        {
            continue;
        }
        // 1. Toan hang (Gia su la chu cai)
        else if ((token >= 'A' && token <= 'Z') ||
                 (token >= 'a' && token <= 'z'))
        {
            postfix_output.push_back(token);
        }
        // 2. Dau mo ngoac
        else if (token == '(')
        {
            stack.push(token);
        }
        // 3. Dau dong ngoac
        else if (token == ')')
        {
            // Pop toan tu cho den khi gap dau '('
            while (!stack.is_empty() && stack.peek() != '(')
            {
                postfix_output.push_back(stack.pop());
            }
            // Pop dau '(' ra khoi Stack (neu co)
            if (!stack.is_empty() && stack.peek() == '(')
            {
                stack.pop();
            }
        }
        // 4. Toan tu
        else if (operators.find(token) != string::npos)
        {
            // Pop toan tu tu Stack ra Output neu:
            // - Stack khong rong
            // - Dinh Stack khong phai la '('
            // - Do uu tien cua dinh Stack >= Do uu tien toan tu hien tai
            while (!stack.is_empty() &&
                   stack.peek() != '(' &&
                   get_precedence(stack.peek()) >= get_precedence(token))
```

```
            {
                postfix_output.push_back(stack.pop());
            }
            // Day toan tu hien tai vao Stack
            stack.push(token);
        }
    }
    // 5. Ket thuc: Pop tat ca toan tu con lai trong Stack
    while (!stack.is_empty())
    {
        // Neu gap '(' o day, bieu thuc Infix bi sai ngoac
        if (stack.peek() == '(')
        {
            return "Bieu thuc Infix khong hop le (Du dau '(')";
        }
        postfix_output.push_back(stack.pop());
    }
    // Chuyen vector thanh chuoi
    string result;
    for (char c : postfix_output)
    {
        result += c;
    }
    return result;
}
int main()
{
    // ---------------- CHAY THU ----------------
    // Bieu thuc vi du: A + B * C
    string infix1 = "A+B*C";
    // Ket qua mong doi: ABC*+
    cout << "Infix: " << infix1
         << " -> Postfix: " << infix_to_postfix(infix1) << endl;
    // Bieu thuc phuc tap hon: (A + B) * (C - D)
    string infix2 = "(A+B)*(C-D)";
    // Ket qua mong doi: AB+CD-*
    cout << "Infix: " << infix2
         << " -> Postfix: " << infix_to_postfix(infix2) << endl;
    // Bieu thuc co luy thua va ngoac: A * (B + C / D) - E
    string infix3 = "A*(B+C/D)-E";
    // Ket qua mong doi: ABCD/+*E-
    cout << "Infix: " << infix3
         << " -> Postfix: " << infix_to_postfix(infix3) << endl;
    // Truong hop xau (Du ngoac mo)
    string infix4 = "((A+B)*C";
    cout << "Infix: " << infix4
         << " -> Postfix: " << infix_to_postfix(infix4) << endl;
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
Infix: A+B*C -> Postfix: ABC*+
Infix: (A+B)*(C-D) -> Postfix: AB+CD-*
Infix: A*(B+C/D)-E -> Postfix: ABCD/+*E-
Infix: ((A+B)*C -> Postfix: Bieu thuc Infix khong hop le (Du dau '(')
```

2.3. Bài tập 3 (Sử dụng collections.deque): Trong Python, nghiên cứu và cài đặt lại Stack bằng collections.deque và giải thích tại sao nó được coi là hiệu quả hơn list cho các thao tác ở cả hai đầu.

2.3.1. Code Python:

```python
from collections import deque
class DequeStack:
    # Cài đặt Stack bằng collections.deque.
    # Sử dụng 'append' để Push và 'pop' để Pop (thao tác ở cùng một đầu).
    def __init__(self):
        # Sử dụng deque làm bộ lưu trữ
        self.items = deque()
    def is_empty(self):
        # Kiểm tra Stack rỗng.
        return len(self.items) == 0
    def push(self, item):
        # Thêm phần tử vào đỉnh Stack (phải). O(1).
        self.items.append(item)
    def pop(self):
        # Lấy và xóa phần tử ở đỉnh Stack (phải). O(1).
        if self.is_empty():
            raise IndexError("Loi: Stack rong, khong the Pop")
        return self.items.pop()
    def peek(self):
        # Xem phần tử ở đỉnh Stack (phải). O(1).
        if self.is_empty():
            raise IndexError("Loi: Stack rong, khong the Peek")
        # Truy cập phần tử cuối cùng
        return self.items[-1]
# ---------------- CHẠY THỬ ----------------
stack_deque = DequeStack()
stack_deque.push("A")
stack_deque.push("B")
stack_deque.push("C")
print(f"Cac phan tu trong Stack: {list(stack_deque.items)}")
print("Peek:", stack_deque.peek())
print("Pop:", stack_deque.pop())
print("Peek sau Pop:", stack_deque.peek())
print("IsEmpty:", stack_deque.is_empty())
```

=> Kết quả sau khi chạy chương trình:

```
Cac phan tu trong Stack: ['A', 'B', 'C']
Peek: C
Pop: C
Peek sau Pop: B
IsEmpty: False
```

2.3.2. Code C++:

```cpp
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
using namespace std;         // Tiền tố std
class DequeStack
{
    // Cai dat Stack bang deque.
    // Su dung push_back de Push va pop_back de Pop (thao tac o cung mot dau).
private:
    deque<string> items;
public:
```

```
    DequeStack()
    {
        // Su dung deque lam bo luu tru
    }
    bool is_empty()
    {
        // Kiem tra Stack rong.
        return items.empty();
    }
    void push(string item)
    {
        // Them phan tu vao dinh Stack (phai). O(1).
        items.push_back(item);
    }
    string pop()
    {
        // Lay va xoa phan tu o dinh Stack (phai). O(1).
        if (is_empty())
        {
            throw runtime_error("Loi: Stack rong, khong the Pop");
        }
        string top_item = items.back();
        items.pop_back();
        return top_item;
    }
    string peek()
    {
        // Xem phan tu o dinh Stack (phai). O(1).
        if (is_empty())
        {
            throw runtime_error("Loi: Stack rong, khong the Peek");
        }
        return items.back();
    }
    void print_stack()
    {
        // In cac phan tu trong Stack
        cout << "[";
        for (size_t i = 0; i < items.size(); i++)
        {
            cout << items[i];
            if (i < items.size() - 1) cout << ", ";
        }
        cout << "]";
    }
};
int main()
{
    DequeStack stack_deque;
    stack_deque.push("A");
    stack_deque.push("B");
    stack_deque.push("C");
    cout << "Cac phan tu trong Stack: ";
    stack_deque.print_stack();
    cout << endl;
    cout << "Peek: " << stack_deque.peek() << endl;
    cout << "Pop: " << stack_deque.pop() << endl;
    cout << "Peek sau Pop: " << stack_deque.peek() << endl;
```

```
    cout << "IsEmpty: " << (stack_deque.is_empty() ? "True" : "False") << endl;
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
Cac phan tu trong Stack: ['A', 'B', 'C']
Peek: C
Pop: C
Peek sau Pop: B
IsEmpty: False
```

BÀI 23. HÀNG ĐỢI (QUEUE)

I. Mục Tiêu:

- **Kiến thức:** HS hiểu rõ Queue là cấu trúc dữ liệu tuyến tính hoạt động theo nguyên tắc **FIFO (First-In, First-Out)**. Nắm vững các thao tác cơ bản: Enqueue, Dequeue, Front/Peek, isEmpty.
- **Kỹ năng:** HS cài đặt được Queue trong Python (sử dụng collections.deque) và C++ (sử dụng std::queue hoặc std::deque).
- **Tư duy:** HS so sánh được Queue với Stack (Nguyên tắc LIFO) và nhận biết các trường hợp sử dụng Queue trong thực tế (Ví dụ: Hàng đợi in, quản lý tác vụ trong hệ điều hành, tìm kiếm BFS).

II. Phân Bổ Thời Gian (90 Phút):

Hoạt Động	Nội Dung	Thời gian (Phút)
Ôn tập	Ôn tập Stack (LIFO) và giới thiệu Queue (FIFO)	10
Lý thuyết 1	Queue: Định nghĩa, Nguyên tắc FIFO, So sánh với Stack	15
Thực hành 1	Bài 1. Cài đặt Queue (Sử dụng deque / std::queue)	30
Thực hành 2	Bài 2. Ứng dụng Queue: Xử lý Tác vụ Cơ bản	25
Tổng kết	So sánh, Bài tập về nhà	10
Tổng cộng		**90**

III. Lý Thuyết 1: Queue - Nguyên tắc FIFO (15 phút)

1. Định nghĩa và Nguyên tắc:

- **Queue (Hàng đợi)** là một cấu trúc dữ liệu trừu tượng (ADT) hoạt động theo nguyên tắc: **FIFO (First-In, First-Out)**
- Nguyên tắc này giống như một hàng người xếp hàng: Người đến **trước nhất** sẽ được phục vụ và rời khỏi hàng **trước nhất**.
- Queue có hai đầu hoạt động:
  + **Rear/Back (Đuôi):** Nơi các phần tử mới được **thêm vào** (Enqueue).
  + **Front/Head (Đầu):** Nơi các phần tử cũ nhất được **lấy ra** (Dequeue).

2. Các Thao tác Cơ bản:

Thao tác	Tên gọi khác	Mô tả	Độ phức tạp
Enqueue	push / insert	Thêm một phần tử vào **Đuôi** (Rear) của Queue.	O(1)
Dequeue	pop / remove	Loại bỏ và trả về phần tử ở **Đầu** (Front) của Queue.	O(1)
Front	peek / top	Chỉ xem và trả về phần tử ở **Đầu** (Front) mà không loại bỏ nó.	O(1)
isEmpty		Kiểm tra xem Queue có rỗng hay không.	O(1)

3. So sánh Stack và Queue:

Cấu trúc	Nguyên tắc	Vị trí Thao tác	Ứng dụng Tiêu biểu
Stack	LIFO (Last-In, First-Out)	Chỉ ở **Đỉnh** (Top)	Undo/Redo, Kiểm tra cân bằng ngoặc
Queue	FIFO (First-In, First-Out)	**Đầu** (Dequeue) và **Đuôi** (Enqueue)	Hàng đợi tác vụ, Thuật toán BFS (Tìm kiếm theo chiều rộng)

IV. Thực Hành 1: Cài đặt Queue (30 phút)

Việc cài đặt Queue đòi hỏi thao tác hiệu quả ở **cả hai đầu** (thêm vào sau, xóa ở trước). Vì lý do này, ta nên sử dụng các cấu trúc hỗ trợ O(1) ở cả hai đầu, như **collections.deque** trong Python và **std::queue (hoặc std::deque)** trong C++.

1. Bài 1: Cài đặt Queue Class

1.1. Code Python:

```python
from collections import deque
class Queue:
    # Cài đặt cấu trúc dữ liệu Queue (Hàng đợi) theo nguyên tắc FIFO
    # sử dụng collections.deque để đảm bảo hiệu suất O(1) cho Enqueue và Dequeue.
    def __init__(self):
        # Bộ lưu trữ chính, sử dụng deque
        self.items = deque()
    def is_empty(self):
        # Kiểm tra xem Queue có rỗng hay không. O(1).
        return not self.items
    def enqueue(self, item):
        # Thêm phần tử vào Đuôi (Rear) của Queue.
        # Thao tác 'append' ở cuối deque là O(1).
        self.items.append(item)
    def dequeue(self):
        # Xóa và trả về phần tử ở Đầu (Front) của Queue.
        # Thao tác 'popleft' ở đầu deque là O(1).
        if self.is_empty():
            raise IndexError("Loi: Queue rong, khong the Dequeue")
        return self.items.popleft() # Lấy ra từ bên trái (Đầu hàng)
    def front(self):
        # Xem phần tử ở Đầu (Front) của Queue mà không xóa. O(1).
        if self.is_empty():
            raise IndexError("Loi: Queue rong, khong the Front")
        return self.items[0] # Truy cập phần tử đầu tiên
# ----------------- CHẠY THỬ VÀ MINH HỌA FIFO -----------------
```

```python
q = Queue()
print("--- THAO TAC ENQUEUE (THEM VAO DUOI) ---")
q.enqueue("Task A") # Task A vào trước nhất
print(f"ENQUEUE: Task A. Hang doi: {list(q.items)}")
q.enqueue("Task B")
print(f"ENQUEUE: Task B. Hang doi: {list(q.items)}")
q.enqueue("Task C") # Task C vào cuối cùng
print(f"ENQUEUE: Task C. Hang doi: {list(q.items)}")
print("\n--- THAO TAC DEQUEUE (LAY RA TU DAU - FIFO) ---")
# Lần 1: Lấy Task A ra (Task A vào trước nhất, nên ra trước nhất)
print(f"FRONT (Dau hang): {q.front()}")
dequeued_item = q.dequeue()
print(f"DEQUEUE: {dequeued_item}. Hang doi sau: {list(q.items)}")
# Lần 2: Lấy Task B ra
dequeued_item = q.dequeue()
print(f"DEQUEUE: {dequeued_item}. Hang doi sau: {list(q.items)}")
# Lần 3: Lấy Task C ra
dequeued_item = q.dequeue()
print(f"DEQUEUE: {dequeued_item}. Hang doi sau: {list(q.items)}")
print("\n--- KIEM TRA ---")
print("IsEmpty (Kiem tra rong):", q.is_empty())
```

=> Kết quả sau khi chạy chương trình:

```
--- THAO TAC ENQUEUE (THEM VAO DUOI) ---
ENQUEUE: Task A. Hang doi: ['Task A']
ENQUEUE: Task B. Hang doi: ['Task A', 'Task B']
ENQUEUE: Task C. Hang doi: ['Task A', 'Task B', 'Task C']

--- THAO TAC DEQUEUE (LAY RA TU DAU - FIFO) ---
FRONT (Dau hang): Task A
DEQUEUE: Task A. Hang doi sau: ['Task B', 'Task C']
DEQUEUE: Task B. Hang doi sau: ['Task C']
DEQUEUE: Task C. Hang doi sau: []

--- KIEM TRA ---
IsEmpty (Kiem tra rong): True
```

- Giải thích Code Python:

deque.popleft(): Đây là hàm then chốt. Nó thực hiện thao tác xóa ở đầu Queue (Front) với độ phức tạp O(1), điều mà list.pop(0) không thể làm được (vì list.pop(0) là O(N)).

1.2. Code C++:

```cpp
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
#include <deque>             // Thư viện xử lí hàng đợi (deque)
#include <string>            // Thư viện xử lí xâu
using namespace std;         // Tiền tố std
class Queue
{
    // Cai dat cau truc du lieu Queue (Hang doi) theo nguyen tac FIFO
    // su dung deque de dam bao hieu suat O(1) cho Enqueue va Dequeue.
private:
    // Bo luu tru chinh, su dung deque
    deque<string> items;
public:
    Queue()
    {
        // Ham khoi tao
    }
```

```
    bool is_empty()
    {
        // Kiem tra xem Queue co rong hay khong. O(1).
        return items.empty();
    }
    void enqueue(string item)
    {
        // Them phan tu vao Duoi (Rear) cua Queue.
        // Thao tac push_back o cuoi deque la O(1).
        items.push_back(item);
    }
    string dequeue()
    {
        // Xoa va tra ve phan tu o Dau (Front) cua Queue.
        // Thao tac pop_front o dau deque la O(1).
        if (is_empty())
        {
            throw runtime_error("Loi: Queue rong, khong the Dequeue");
        }
        string front_item = items.front(); // Lay phan tu dau hang
        items.pop_front();                 // Xoa phan tu dau hang
        return front_item;
    }
    string front()
    {
        // Xem phan tu o Dau (Front) cua Queue ma khong xoa. O(1).
        if (is_empty())
        {
            throw runtime_error("Loi: Queue rong, khong the Front");
        }
        return items.front(); // Truy cap phan tu dau tien
    }
    void print_queue()
    {
        // In cac phan tu trong Queue
        cout << "[";
        for (size_t i = 0; i < items.size(); i++)
        {
            cout << items[i];
            if (i < items.size() - 1) cout << ", ";
        }
        cout << "]";
    }
};
int main()
{
    // ---------------- CHAY THU VA MINH HOA FIFO ----------------
    Queue q;
    cout << "--- THAO TAC ENQUEUE (THEM VAO DUOI) ---" << endl;
    q.enqueue("Task A"); // Task A vao truoc nhat
    cout << "ENQUEUE: Task A. Hang doi: ";
    q.print_queue();
    cout << endl;
    q.enqueue("Task B");
    cout << "ENQUEUE: Task B. Hang doi: ";
    q.print_queue();
    cout << endl;
    q.enqueue("Task C"); // Task C vao cuoi cung
```

```
    cout << "ENQUEUE: Task C. Hang doi: ";
    q.print_queue();
    cout << endl;
    cout << "\n--- THAO TAC DEQUEUE (LAY RA TU DAU - FIFO) ---" << endl;
    // Lan 1: Lay Task A ra (Task A vao truoc nhat, nen ra truoc nhat)
    cout << "FRONT (Dau hang): " << q.front() << endl;
    string dequeued_item = q.dequeue();
    cout << "DEQUEUE: " << dequeued_item << ". Hang doi sau: ";
    q.print_queue();
    cout << endl;
    // Lan 2: Lay Task B ra
    dequeued_item = q.dequeue();
    cout << "DEQUEUE: " << dequeued_item << ". Hang doi sau: ";
    q.print_queue();
    cout << endl;
    // Lan 3: Lay Task C ra
    dequeued_item = q.dequeue();
    cout << "DEQUEUE: " << dequeued_item << ". Hang doi sau: ";
    q.print_queue();
    cout << endl;
    cout << "\n--- KIEM TRA ---" << endl;
    cout << "IsEmpty (Kiem tra rong): "
         << (q.is_empty() ? "True" : "False") << endl;
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
--- THAO TAC ENQUEUE (THEM VAO DUOI) ---
ENQUEUE: Task A. Hang doi: ['Task A']
ENQUEUE: Task B. Hang doi: ['Task A', 'Task B']
ENQUEUE: Task C. Hang doi: ['Task A', 'Task B', 'Task C']

--- THAO TAC DEQUEUE (LAY RA TU DAU - FIFO) ---
FRONT (Dau hang): Task A
DEQUEUE: Task A. Hang doi sau: ['Task B', 'Task C']
DEQUEUE: Task B. Hang doi sau: ['Task C']
DEQUEUE: Task C. Hang doi sau: []

--- KIEM TRA ---
IsEmpty (Kiem tra rong): True
```

- Giải thích Code C++:

std::queue: C++ cung cấp **container adapter std::queue**. Các thao tác tương ứng là push() (enqueue), pop() (xóa đầu), và front() (xem đầu). Giống như **deque Python**, nó đảm bảo hiệu suất O(1).

V. Thực Hành 2: Ứng dụng Queue (25 phút)

Ứng dụng cơ bản nhất của Queue là mô phỏng **Xử lý tác vụ/yêu cầu** theo thứ tự đến trước.

1. Bài 2: Mô phỏng Hàng đợi In (Print Queue)

Ta sẽ mô phỏng một hệ thống in ấn nơi các tài liệu được in theo thứ tự chúng được gửi đến.

1.1. Code Python:

```
from collections import deque
# --- ĐỊNH NGHĨA LỚP QUEUE (Cần thiết cho hàm process_print_jobs) ---
class Queue:
    # Cấu trúc dữ liệu Queue sử dụng collections.deque.
```

```
    def __init__(self):
        self.items = deque()
    def is_empty(self):
        return not self.items
    def enqueue(self, item):
        # Thêm vào Đuôi (phải)
        self.items.append(item)
    def dequeue(self):
        # Xóa và trả về phần tử ở Đầu (trái)
        if self.is_empty():
            raise IndexError("Loi: Queue rong, khong the Dequeue")
        return self.items.popleft()
    # Hàm front được thêm vào để hoàn thiện lớp Queue
    def front(self):
        if self.is_empty():
            raise IndexError("Loi: Queue rong, khong the Front")
        return self.items[0]
# --- HÀM MÔ PHỎNG XỬ LÝ TÁC VỤ IN ---
def process_print_jobs(jobs):
    # Mô phỏng xử lý hàng đợi in, tuân theo nguyên tắc FIFO.
    # Tham số:
    # jobs (list): Danh sách các tên file cần in.
    print_queue = Queue()
    print("-" * 45)
    print("--- BUOC 1: ENQUEUE (THEM VAO HANG DOI) ---")
    # 1. Enqueue: Thêm các tác vụ vào hàng đợi theo thứ tự đến
    for job in jobs:
        print(f"[THEM] File '{job}' duoc them vao hang doi.")
        print_queue.enqueue(job)
    print("-" * 45)
    print("--- BUOC 2: DEQUEUE (XU LY - FIFO) ---")
    print("BAT DAU IN AN (First-In, First-Out)")
    # 2. Dequeue: Xử lý các tác vụ theo thứ tự FIFO
    while not print_queue.is_empty():
        current_job = print_queue.dequeue()
        print(f"[IN AN] Dang xu ly file: {current_job}")
    print("-" * 45)
    print("Tat ca tac vu da hoan thanh.")
# ----------------- CHẠY THỬ -----------------
jobs_list = ["BaoCao_Quy1.pdf", "Anh_KyNiem.jpg", "TaiLieu_HocTap.docx"]
print(">>> KICH BAN: HE THONG IN AN <<<")
process_print_jobs(jobs_list)
```

=> Kết quả sau khi chạy chương trình:

```
>>> KICH BAN: HE THONG IN AN <<<
---------------------------------------------
--- BUOC 1: ENQUEUE (THEM VAO HANG DOI) ---
[THEM] File 'BaoCao_Quy1.pdf' duoc them vao hang doi.
[THEM] File 'Anh_KyNiem.jpg' duoc them vao hang doi.
[THEM] File 'TaiLieu_HocTap.docx' duoc them vao hang doi.
---------------------------------------------
--- BUOC 2: DEQUEUE (XU LY - FIFO) ---
BAT DAU IN AN (First-In, First-Out)
[IN AN] Dang xu ly file: BaoCao_Quy1.pdf
[IN AN] Dang xu ly file: Anh_KyNiem.jpg
[IN AN] Dang xu ly file: TaiLieu_HocTap.docx
---------------------------------------------
Tat ca tac vu da hoan thanh.
```

- Giải thích Code Python:

Thứ tự thực hiện: Tác vụ được thêm vào đầu tiên (BaoCao_Quy1.pdf) phải được xử lý (Dequeue) đầu tiên, đảm bảo tính công bằng theo thời gian đến.

1.2. Code C++:

```
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
#include <deque>             // Thư viện xử lí hàng đợi (deque)
#include <string>            // Thư viện xử lí xâu
using namespace std;         // Tiền tố std
// --- DINH NGHIA LOP QUEUE (Can thiet cho ham process_print_jobs) ---
class Queue
{
    // Cau truc du lieu Queue su dung deque.
private:
    deque<string> items;
public:
    Queue()
    {
        // Ham khoi tao
    }
    bool is_empty()
    {
        return items.empty();
    }
    void enqueue(string item)
    {
        // Them vao Duoi (phai)
        items.push_back(item);
    }
    string dequeue()
    {
        // Xoa va tra ve phan tu o Dau (trai)
        if (is_empty())
        {
            throw runtime_error("Loi: Queue rong, khong the Dequeue");
        }
        string front_item = items.front();
        items.pop_front();
        return front_item;
    }
    // Ham front duoc them vao de hoan thien lop Queue
    string front()
    {
        if (is_empty())
        {
            throw runtime_error("Loi: Queue rong, khong the Front");
        }
        return items.front();
    }
};
// --- HAM MO PHONG XU LY TAC VU IN ---
void process_print_jobs(const vector<string>& jobs)
{
    // Mo phong xu ly hang doi in, tuan theo nguyen tac FIFO.
    // Tham so:
    // jobs (vector<string>): Danh sach cac ten file can in.
```

```
    Queue print_queue;
    cout << string(45, '-') << endl;
    cout << "--- BUOC 1: ENQUEUE (THEM VAO HANG DOI) ---" << endl;
    // 1. Enqueue: Them cac tac vu vao hang doi theo thu tu den
    for (const string& job : jobs)
    {
        cout << "[THEM] File '" << job << "' duoc them vao hang doi." << endl;
        print_queue.enqueue(job);
    }
    cout << string(45, '-') << endl;
    cout << "--- BUOC 2: DEQUEUE (XU LY - FIFO) ---" << endl;
    cout << "BAT DAU IN AN (First-In, First-Out)" << endl;
    // 2. Dequeue: Xu ly cac tac vu theo thu tu FIFO
    while (!print_queue.is_empty())
    {
        string current_job = print_queue.dequeue();
        cout << "[IN AN] Dang xu ly file: " << current_job << endl;
    }
    cout << string(45, '-') << endl;
    cout << "Tat ca tac vu da hoan thanh." << endl;
}
// ----------------- CHAY THU -----------------
int main()
{
    vector<string> jobs_list = {
        "BaoCao_Quy1.pdf",
        "Anh_KyNiem.jpg",
        "TaiLieu_HocTap.docx"
    };
    cout << ">>> KICH BAN: HE THONG IN AN <<<" << endl;
    process_print_jobs(jobs_list);
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
>>> KICH BAN: HE THONG IN AN <<<
---------------------------------------------
--- BUOC 1: ENQUEUE (THEM VAO HANG DOI) ---
[THEM] File 'BaoCao_Quy1.pdf' duoc them vao hang doi.
[THEM] File 'Anh_KyNiem.jpg' duoc them vao hang doi.
[THEM] File 'TaiLieu_HocTap.docx' duoc them vao hang doi.
---------------------------------------------
--- BUOC 2: DEQUEUE (XU LY - FIFO) ---
BAT DAU IN AN (First-In, First-Out)
[IN AN] Dang xu ly file: BaoCao_Quy1.pdf
[IN AN] Dang xu ly file: Anh_KyNiem.jpg
[IN AN] Dang xu ly file: TaiLieu_HocTap.docx
---------------------------------------------
Tat ca tac vu da hoan thanh.
```

- Giải thích Code C++:

Tách biệt front() và pop(): Trong C++, việc lấy giá trị (front()) và xóa phần tử (pop()) là hai thao tác riêng biệt, cần phải thực hiện theo thứ tự.

VI. Tổng kết & Bài tập Về nhà (10 phút)

1. Tóm tắt:

Queue là cấu trúc dữ liệu mô phỏng hàng đợi trong thế giới thực (FIFO). Các thao tác thêm (Enqueue) và xóa (Dequeue) đều được thực hiện ở các đầu khác nhau với hiệu suất O(1), làm cho Queue trở thành công cụ không thể thiếu trong các hệ thống xử lý tác vụ tuần tự và các thuật toán tìm kiếm.

2. Bài tập Về nhà:

2.1. Bài tập 1 (Queue Vòng tròn): Cài đặt **Circular Queue (Queue Vòng tròn)** sử dụng một mảng có kích thước cố định. Giải thích lợi ích của nó so với Queue tuyến tính truyền thống.

2.1.1. Code Python:

```
class CircularQueue:
    # Cài đặt Circular Queue sử dụng mảng/list có kích thước cố định.
    # Sử dụng (rear + 1) % capacity để phân biệt trạng thái đầy/rỗng.
    def __init__(self, capacity):
        # Kích thước tối đa của Queue (số phần tử có thể lưu trữ)
        self.capacity = capacity
        # Mảng/list dùng để lưu trữ dữ liệu. +1 là cho phần tử dự phòng
        self.items = [None] * (capacity + 1)
        # Chỉ mục Đầu (Front) và Đuôi (Rear).
        # Cả hai khởi tạo ở vị trí 0
        self.front = 0
        self.rear = 0
    def is_empty(self):
        # Kiểm tra Queue rỗng: khi Front và Rear trùng nhau.
        return self.front == self.rear
    def is_full(self):
        # Kiểm tra Queue đầy: khi (rear + 1) % (capacity + 1) == front.
        # Chúng ta sử dụng một slot dự phòng, vì vậy kích thước thực sự là
capacity.
        return (self.rear + 1) % (self.capacity + 1) == self.front
    def enqueue(self, item):
        # Thêm phần tử vào đuôi Queue (Rear). O(1).
        if self.is_full():
            print("Loi: Queue da day, khong the Enqueue.")
            return False
        # Tăng Rear và dùng phép toán modulo để cuộn vòng
        self.rear = (self.rear + 1) % (self.capacity + 1)
        self.items[self.rear] = item
        print(f"  [ENQUEUE] '{item}' tai rear={self.rear}")
        return True
    def dequeue(self):
        # Xóa và trả về phần tử ở đầu Queue (Front). O(1).
        if self.is_empty():
            raise IndexError("Loi: Queue rong, khong the Dequeue.")
        # Tăng Front và dùng phép toán modulo để cuộn vòng
        self.front = (self.front + 1) % (self.capacity + 1)
        data = self.items[self.front]
        self.items[self.front] = None  # Đánh dấu là trống
        print(f"  [DEQUEUE] '{data}' tu front={self.front}")
        return data
    def size(self):
        # Tính kích thước hiện tại của Queue.
        # Sử dụng phép toán modulo để tính toán chính xác số phần tử
        return (self.rear - self.front + self.capacity + 1) % (self.capacity + 1)
# ---------------- CHẠY THỬ VÀ MINH HỌA VÒNG TRÒN ----------------
MAX_CAPACITY = 3
```

```python
cq = CircularQueue(MAX_CAPACITY) # Kích thước 3, mảng vật lý 4 slots
# 1. Enqueue 3 phần tử
cq.enqueue(10) # rear=1
cq.enqueue(20) # rear=2
cq.enqueue(30) # rear=3
print(f"Trang thai (Size={cq.size()}): {cq.items}")
# 2. Dequeue 2 phần tử (Tạo khoảng trống ở đầu)
cq.dequeue() # front=1
cq.dequeue() # front=2
print(f"Trang thai sau Dequeue (Size={cq.size()}): {cq.items}")
# 3. Enqueue thêm 2 phần tử mới (Rear sẽ quay lại vị trí đầu mảng)
print("\n--- BAT DAU VONG LAP (WRAP AROUND) ---")
cq.enqueue(40) # rear=0 (Quay lại đầu mảng!)
print(f"Trang thai sau ENQUEUE 40 (Size={cq.size()}): {cq.items}")
# Thử thêm phần tử cuối cùng
cq.enqueue(50) # rear=1
print(f"Trang thai sau ENQUEUE 50 (Size={cq.size()}): {cq.items}")
# 4. Kiểm tra trạng thái Đầy
print(f"\nKiam tra day: {cq.is_full()}") # Kết quả: True
cq.enqueue(60) # Sẽ báo lỗi đầy
```

=> Kết quả sau khi chạy chương trình:

```
  [ENQUEUE] '10' tai rear=1
  [ENQUEUE] '20' tai rear=2
  [ENQUEUE] '30' tai rear=3
Trang thai (Size=3): [None, 10, 20, 30]
  [DEQUEUE] '10' tu front=1
  [DEQUEUE] '20' tu front=2
Trang thai sau Dequeue (Size=1): [None, None, None, 30]

--- BAT DAU VONG LAP (WRAP AROUND) ---
  [ENQUEUE] '40' tai rear=0
Trang thai sau ENQUEUE 40 (Size=2): [40, None, None, 30]
  [ENQUEUE] '50' tai rear=1
Trang thai sau ENQUEUE 50 (Size=3): [40, 50, None, 30]

Kiam tra day: True
Loi: Queue da day, khong the Enqueue.
```

2.1.2. Code C++:

```cpp
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
#include <deque>             // Thư viện xử lí hàng đợi (deque)
#include <string>            // Thư viện xử lí xâu
using namespace std;         // Tiền tố std
class CircularQueue
{
    // Cai dat Circular Queue su dung mang co kich thuoc co dinh.
    // Su dung (rear + 1) % capacity de phan biet trang thai day/rong.
private:
    int capacity;                // Kich thuoc toi da cua Queue
    vector<int> items;           // Mang luu tru du lieu (+1 slot du phong)
    int front;                   // Chi muc Dau
    int rear;                    // Chi muc Duoi
public:
    CircularQueue(int capacity)
    {
        // Kich thuoc toi da cua Queue (so phan tu co the luu tru)
```

```
        this->capacity = capacity;
        // Mang dung de luu tru du lieu. +1 la cho phan tu du phong
        items.resize(capacity + 1, -1);
        // Ca hai khoi tao o vi tri 0
        front = 0;
        rear = 0;
    }
    bool is_empty()
    {
        // Kiem tra Queue rong: khi Front va Rear trung nhau
        return front == rear;
    }
    bool is_full()
    {
        // Kiem tra Queue day:
        // (rear + 1) % (capacity + 1) == front
        return (rear + 1) % (capacity + 1) == front;
    }
    bool enqueue(int item)
    {
        // Them phan tu vao duoi Queue (Rear). O(1)
        if (is_full())
        {
            cout << "Loi: Queue da day, khong the Enqueue." << endl;
            return false;
        }
        // Tang Rear va dung phep toan modulo de cuon vong
        rear = (rear + 1) % (capacity + 1);
        items[rear] = item;
        cout << "  [ENQUEUE] '" << item << "' tai rear=" << rear << endl;
        return true;
    }
    int dequeue()
    {
        // Xoa va tra ve phan tu o dau Queue (Front). O(1)
        if (is_empty())
        {
            throw runtime_error("Loi: Queue rong, khong the Dequeue.");
        }
        // Tang Front va dung phep toan modulo de cuon vong
        front = (front + 1) % (capacity + 1);
        int data = items[front];
        items[front] = -1; // Danh dau la trong
        cout << "  [DEQUEUE] '" << data << "' tu front=" << front << endl;
        return data;
    }
    int size()
    {
        // Tinh kich thuoc hien tai cua Queue
        return (rear - front + capacity + 1) % (capacity + 1);
    }
    void print_state()
    {
        // In trang thai mang ben trong
        cout << "[ ";
        for (int i = 0; i < items.size(); i++)
        {
            if (items[i] == -1)
```

```
                cout << "None ";
            else
                cout << items[i] << " ";
        }
        cout << "]" << endl;
    }
};
// ----------------- CHAY THU VA MINH HOA VONG TRON -----------------
int main()
{
    int MAX_CAPACITY = 3;
    CircularQueue cq(MAX_CAPACITY); // Kich thuoc 3, mang vat ly 4 slots
    // 1. Enqueue 3 phan tu
    cq.enqueue(10); // rear=1
    cq.enqueue(20); // rear=2
    cq.enqueue(30); // rear=3
    cout << "Trang thai (Size=" << cq.size() << "): ";
    cq.print_state();
    // 2. Dequeue 2 phan tu (Tao khoang trong o dau)
    cq.dequeue(); // front=1
    cq.dequeue(); // front=2
    cout << "Trang thai sau Dequeue (Size=" << cq.size() << "): ";
    cq.print_state();
    // 3. Enqueue them 2 phan tu moi (Rear quay vong ve dau mang)
    cout << "\n--- BAT DAU VONG LAP (WRAP AROUND) ---" << endl;
    cq.enqueue(40); // rear=0
    cout << "Trang thai sau ENQUEUE 40 (Size=" << cq.size() << "): ";
    cq.print_state();
    // Thu them phan tu cuoi cung
    cq.enqueue(50); // rear=1
    cout << "Trang thai sau ENQUEUE 50 (Size=" << cq.size() << "): ";
    cq.print_state();
    // 4. Kiem tra trang thai day
    cout << "\nKiem tra day: " << (cq.is_full() ? "True" : "False") << endl;
    cq.enqueue(60); // Se bao loi day
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
  [ENQUEUE] '10' tai rear=1
  [ENQUEUE] '20' tai rear=2
  [ENQUEUE] '30' tai rear=3
Trang thai (Size=3): [None, 10, 20, 30]
  [DEQUEUE] '10' tu front=1
  [DEQUEUE] '20' tu front=2
Trang thai sau Dequeue (Size=1): [None, None, None, 30]

--- BAT DAU VONG LAP (WRAP AROUND) ---
  [ENQUEUE] '40' tai rear=0
Trang thai sau ENQUEUE 40 (Size=2): [40, None, None, 30]
  [ENQUEUE] '50' tai rear=1
Trang thai sau ENQUEUE 50 (Size=3): [40, 50, None, 30]

Kiam tra day: True
Loi: Queue da day, khong the Enqueue.
```

2.2. Bài tập 2 (Deque): Cài đặt **Deque (Double-Ended Queue)**, một biến thể của Queue cho phép enqueue và dequeue ở cả hai đầu. (Lưu ý: Có thể sử dụng collections.deque và định nghĩa các hàm add_first, add_last, remove_first, remove_last).

2.2.1. Code Python:

```
from collections import deque
class Deque:
    # Cài đặt Double-Ended Queue (Deque) sử dụng collections.deque.
    # Hỗ trợ thêm/xóa ở cả hai đầu với hiệu suất O(1).
    def __init__(self):
        self._items = deque()
    def is_empty(self):
        # Kiểm tra Deque rỗng.
        return len(self._items) == 0
    def add_first(self, item):
        # Thêm phần tử vào Đầu (Front). O(1).
        # Sử dụng deque.appendleft().
        self._items.appendleft(item)
    def add_last(self, item):
        # Thêm phần tử vào Đuôi (Rear). O(1).
        # Sử dụng deque.append().
        self._items.append(item)
    def remove_first(self):
        # Xóa và trả về phần tử ở Đầu (Front). O(1).
        # Sử dụng deque.popleft().
        if self.is_empty():
            raise IndexError("Loi: Deque rong, khong the remove_first")
        return self._items.popleft()
    def remove_last(self):
        # Xóa và trả về phần tử ở Đuôi (Rear). O(1).
        # Sử dụng deque.pop().
        if self.is_empty():
            raise IndexError("Loi: Deque rong, khong the remove_last")
        return self._items.pop()
    def size(self):
        # Trả về số lượng phần tử.
        return len(self._items)
    def display(self):
        # Hiển thị Deque để theo dõi.
        return list(self._items)
# ----------------- CHẠY THỬ VÀ MINH HỌA -----------------
d = Deque()
print("--- THAO TAC THEM ---")
d.add_last(10)      # Thêm 10 vào cuối [10]
d.add_first(5)      # Thêm 5 vào đầu [5, 10]
d.add_last(20)      # Thêm 20 vào cuối [5, 10, 20]
d.add_first(1)      # Thêm 1 vào đầu [1, 5, 10, 20]
print(f"Trang thai Deque hien tai: {d.display()}")
print(f"Kich thuoc: {d.size()}")
print("\n--- THAO TAC XOA (CA HAI DAU) ---")
# Xóa ở Đầu (Front)
removed_first = d.remove_first() # Lấy 1
print(f"Removed First (Front): {removed_first}. Deque: {d.display()}")
# Xóa ở Đuôi (Rear)
removed_last = d.remove_last() # Lấy 20
print(f"Removed Last (Rear): {removed_last}. Deque: {d.display()}")
```

```
# Xóa tiếp (Chỉ còn [5, 10])
d.remove_first()
d.remove_last()
print("\n--- KIEM TRA RONG ---")
print(f"Deque rong? {d.is_empty()}") # Kết quả: True
# Thử xóa khi rỗng (Sẽ báo lỗi IndexError)
# d.remove_first()
```

=> Kết quả sau khi chạy chương trình:

```
--- THAO TAC THEM ---
Trang thai Deque hien tai: [1, 5, 10, 20]
Kich thuoc: 4

--- THAO TAC XOA (CA HAI DAU) ---
Removed First (Front): 1. Deque: [5, 10, 20]
Removed Last (Rear): 20. Deque: [5, 10]

--- KIEM TRA RONG ---
Deque rong? True
```

2.2.2. Code C++:

```
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
#include <deque>             // Thư viện xử lí hàng đợi (deque)
#include <string>            // Thư viện xử lí xâu
#include <stdexcept>         // Thư viện báo lỗi
using namespace std;         // Tiền tố std
class Deque
{
    // Cai dat Double-Ended Queue (Deque) su dung STL deque.
    // Ho tro them/xoa o ca hai dau voi hieu suat O(1).
private:
    deque<int> _items;   // Bo luu tru chinh
public:
    Deque()
    {
        // Ham khoi tao
    }
    bool is_empty()
    {
        // Kiem tra Deque rong
        return _items.empty();
    }
    void add_first(int item)
    {
        // Them phan tu vao Dau (Front). O(1)
        // Su dung deque.push_front()
        _items.push_front(item);
    }
    void add_last(int item)
    {
        // Them phan tu vao Duoi (Rear). O(1)
        // Su dung deque.push_back()
        _items.push_back(item);
    }
    int remove_first()
    {
        // Xoa va tra ve phan tu o Dau (Front). O(1)
```

```
        // Su dung deque.pop_front()
        if (is_empty())
        {
            throw runtime_error("Loi: Deque rong, khong the remove_first");
        }
        int value = _items.front();
        _items.pop_front();
        return value;
    }
    int remove_last()
    {
        // Xoa va tra ve phan tu o Duoi (Rear). O(1)
        // Su dung deque.pop_back()
        if (is_empty())
        {
            throw runtime_error("Loi: Deque rong, khong the remove_last");
        }
        int value = _items.back();
        _items.pop_back();
        return value;
    }
    int size()
    {
        // Tra ve so luong phan tu
        return _items.size();
    }
    void display()
    {
        // Hien thi Deque de theo doi
        cout << "[";
        for (int i = 0; i < _items.size(); i++)
        {
            cout << _items[i];
            if (i < _items.size() - 1) cout << ", ";
        }
        cout << "]" << endl;
    }
};
// ----------------- CHAY THU VA MINH HOA -----------------
int main()
{
    Deque d;
    cout << "--- THAO TAC THEM ---" << endl;
    d.add_last(10);      // Them 10 vao cuoi [10]
    d.add_first(5);      // Them 5 vao dau [5, 10]
    d.add_last(20);      // Them 20 vao cuoi [5, 10, 20]
    d.add_first(1);      // Them 1 vao dau [1, 5, 10, 20]
    cout << "Trang thai Deque hien tai: ";
    d.display();
    cout << "Kich thuoc: " << d.size() << endl;
    cout << "\n--- THAO TAC XOA (CA HAI DAU) ---" << endl;
    // Xoa o Dau (Front)
    int removed_first = d.remove_first(); // Lay 1
    cout << "Removed First (Front): " << removed_first << ". Deque: ";
    d.display();
    // Xoa o Duoi (Rear)
    int removed_last = d.remove_last(); // Lay 20
    cout << "Removed Last (Rear): " << removed_last << ". Deque: ";
```

```
    d.display();
    // Xoa tiep (Chi con [5, 10])
    d.remove_first();
    d.remove_last();
    cout << "\n--- KIEM TRA RONG ---" << endl;
    cout << "Deque rong? " << (d.is_empty() ? "True" : "False") << endl;
    // Thu xoa khi rong (Se bao loi runtime_error)
    // d.remove_first();
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
--- THAO TAC THEM ---
Trang thai Deque hien tai: [1, 5, 10, 20]
Kich thuoc: 4

--- THAO TAC XOA (CA HAI DAU) ---
Removed First (Front): 1. Deque: [5, 10, 20]
Removed Last (Rear): 20. Deque: [5, 10]

--- KIEM TRA RONG ---
Deque rong? True
```

2.3. Bài tập 3 (Sử dụng Queue): Viết chương trình sử dụng Queue để mô phỏng **Quay số may mắn** (tìm người thắng cuộc trong một vòng tròn) theo thuật toán Josephus.

2.3.1. Code Python:

```python
from collections import deque
class Queue:
    # Cấu trúc dữ liệu Queue cơ bản sử dụng collections.deque.
    def __init__(self):
        self.items = deque()
    def is_empty(self):
        return not self.items
    def enqueue(self, item):
        self.items.append(item)
    def dequeue(self):
        if self.is_empty():
            raise IndexError("Queue is empty")
        return self.items.popleft()
    def size(self):
        return len(self.items)
def solve_josephus_problem(num_people, k_step):
    # Sử dụng Queue để mô phỏng vấn đề Josephus và tìm ra người thắng cuộc.
    # Tham số:
    #   num_people (int): Tổng số người tham gia (N).
    #   k_step (int): Bước nhảy để loại bỏ (K).
    # Trả về:
    #   int: Số của người chiến thắng.
    if num_people <= 0 or k_step <= 0:
        return None
    # 1. Khởi tạo: Đẩy N người vào Queue (đánh số từ 1 đến N)
    queue = Queue()
    for i in range(1, num_people + 1):
        queue.enqueue(i)
    print(f"Tong so nguoi: {num_people}, Buoc nhay (K): {k_step}")
    print(f"Bat dau vong quay: {list(queue.items)}")
    print("-" * 40)
    # 2. Lặp vòng: Tiếp tục cho đến khi chỉ còn 1 người
```

```
    while queue.size() > 1:
        # 3. Bỏ qua (K-1 lần): Dequeue và Enqueue lại
        for _ in range(k_step - 1):
            person_to_move = queue.dequeue()
            queue.enqueue(person_to_move)
            # print(f"  Bỏ qua: {person_to_move}. Queue: {list(queue.items)}")
        # 4. Loại bỏ (Lần thứ K): Dequeue và không Enqueue lại
        person_eliminated = queue.dequeue()
        print(f"  [LOAI BO] Nguoi so {person_eliminated}. Queue con:
{list(queue.items)}")
    # Người còn lại trong Queue là người chiến thắng
    winner = queue.dequeue()
    return winner
# ----------------- CHẠY THỬ -----------------
N = 7  # 7 người
K = 3  # Cứ 3 bước thì loại bỏ 1 người
winner_id = solve_josephus_problem(N, K)
print("-" * 40)
if winner_id is not None:
    print(f"Nguoi chien thang la SO: {winner_id}")
else:
    print("Thong so khong hop le.")
```

=> Kết quả sau khi chạy chương trình:

```
Tong so nguoi: 7, Buoc nhay (K): 3
Bat dau vong quay: [1, 2, 3, 4, 5, 6, 7]
----------------------------------------
  [LOAI BO] Nguoi so 3. Queue con: [4, 5, 6, 7, 1, 2]
  [LOAI BO] Nguoi so 6. Queue con: [7, 1, 2, 4, 5]
  [LOAI BO] Nguoi so 2. Queue con: [4, 5, 7, 1]
  [LOAI BO] Nguoi so 7. Queue con: [1, 4, 5]
  [LOAI BO] Nguoi so 5. Queue con: [1, 4]
  [LOAI BO] Nguoi so 1. Queue con: [4]
----------------------------------------
Nguoi chien thang la SO: 4
```

2.3.2. Code C++:

```
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
#include <deque>             // Thư viện xử lí hàng đợi (deque)
using namespace std;         // Tiền tố std
class Queue
{
    // Cau truc du lieu Queue co ban su dung deque.
private:
    deque<int> items;
public:
    Queue()
    {
        // Ham khoi tao
    }
    bool is_empty()
    {
        return items.empty();
    }
    void enqueue(int item)
    {
        items.push_back(item);
```

```
    }
    int dequeue()
    {
        if (is_empty())
        {
            throw runtime_error("Queue is empty");
        }
        int front_item = items.front();
        items.pop_front();
        return front_item;
    }
    int size()
    {
        return items.size();
    }
    // Ham phu de hien thi Queue (tuong duong list(queue.items) trong Python)
    void display()
    {
        cout << "[";
        for (size_t i = 0; i < items.size(); i++)
        {
            cout << items[i];
            if (i < items.size() - 1) cout << ", ";
        }
        cout << "]";
    }
};
int solve_josephus_problem(int num_people, int k_step)
{
    // Su dung Queue de mo phong van de Josephus va tim ra nguoi thang cuoc.
    // Tham so:
    //   num_people (int): Tong so nguoi tham gia (N).
    //   k_step (int): Buoc nhay de loai bo (K).
    // Tra ve:
    //   int: So cua nguoi chien thang.
    if (num_people <= 0 || k_step <= 0)
    {
        return -1;
    }
    // 1. Khoi tao: Day N nguoi vao Queue (danh so tu 1 den N)
    Queue queue;
    for (int i = 1; i <= num_people; i++)
    {
        queue.enqueue(i);
    }
    cout << "Tong so nguoi: " << num_people << ", Buoc nhay (K): " << k_step <<
endl;
    cout << "Bat dau vong quay: ";
    queue.display();
    cout << endl;
    cout << "----------------------------------------" << endl;
    // 2. Lap vong: Tiep tuc cho den khi chi con 1 nguoi
    while (queue.size() > 1)
    {
        // 3. Bo qua (K-1 lan): Dequeue va Enqueue lai
        for (int i = 0; i < k_step - 1; i++)
        {
            int person_to_move = queue.dequeue();
```

```
            queue.enqueue(person_to_move);
        }
        // 4. Loai bo (Lan thu K): Dequeue va khong Enqueue lai
        int person_eliminated = queue.dequeue();
        cout << "  [LOAI BO] Nguoi so " << person_eliminated << ". Queue con: ";
        queue.display();
        cout << endl;
    }
    // Nguoi con lai trong Queue la nguoi chien thang
    int winner = queue.dequeue();
    return winner;
}
// ----------------- CHAY THU -----------------
int main()
{
    int N = 7; // 7 nguoi
    int K = 3; // Cu 3 buoc thi loai bo 1 nguoi
    int winner_id = solve_josephus_problem(N, K);
    cout << "----------------------------------------" << endl;
    if (winner_id != -1)
    {
        cout << "Nguoi chien thang la SO: " << winner_id << endl;
    }
    else
    {
        cout << "Thong so khong hop le." << endl;
    }
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
Tong so nguoi: 7, Buoc nhay (K): 3
Bat dau vong quay: [1, 2, 3, 4, 5, 6, 7]
----------------------------------------
  [LOAI BO] Nguoi so 3. Queue con: [4, 5, 6, 7, 1, 2]
  [LOAI BO] Nguoi so 6. Queue con: [7, 1, 2, 4, 5]
  [LOAI BO] Nguoi so 2. Queue con: [4, 5, 7, 1]
  [LOAI BO] Nguoi so 7. Queue con: [1, 4, 5]
  [LOAI BO] Nguoi so 5. Queue con: [1, 4]
  [LOAI BO] Nguoi so 1. Queue con: [4]
----------------------------------------
Nguoi chien thang la SO: 4
```

BÀI 24. MẢNG CỘNG DỒN

I. Mục Tiêu:

- **Kiến thức:** HS hiểu rõ khái niệm Mảng cộng dồn (Prefix Sum Array) và cách nó được xây dựng. Nắm vững ứng dụng chính: Tính tổng đoạn con (range sum query) trong O(1).
- **Kỹ năng:** HS cài đặt được Prefix Sum Array 1D và giải quyết các bài toán tính tổng đoạn con trong Python và C++.
- **Tư duy:** HS phân biệt được độ phức tạp O(N) (dùng vòng lặp) và O(1) (dùng Prefix Sum) khi tính tổng đoạn con.

II. Phân Bổ Thời Gian (90 Phút):

Hoạt Động	Nội Dung	Thời gian (Phút)
Ôn tập	Ôn tập Queue (FIFO) và độ phức tạp O(N) và O(1)	10
Lý thuyết 1	Prefix Sum 1D: Định nghĩa, Xây dựng mảng	15
Thực hành 1	Bài 1. Xây dựng và tính tổng đoạn con (1D)	30
Thực hành 2	Bài 2. Prefix Sum nâng cao: Tìm điểm cân bằng	25
Tổng kết	Mở rộng 2D, Bài tập về nhà	10
Tổng cộng		**90**

III. Lý Thuyết 1: Mảng cộng dồn 1D (Prefix Sum Array) (15 phút)

1. Định nghĩa:

- **Mảng cộng dồn (Prefix Sum Array)**, ký hiệu là P, là một mảng phụ trợ được xây dựng từ mảng gốc A. Mỗi phần tử P[i] lưu trữ **tổng của tất cả các phần tử trong mảng gốc A từ chỉ mục 0 đến chỉ mục i** (bao gồm cả A[i]).
- Công thức định nghĩa: **P[i] = A[0] + A[1] + … + A[i]**

2. Xây dựng Mảng cộng dồn:

- Mảng cộng dồn có thể được xây dựng hiệu quả chỉ trong O(N) (với N là kích thước mảng gốc) bằng cách sử dụng công thức đệ quy đơn giản:
- Công thức đệ quy: **P[i] = P[i-1] + A[i]**

(Với P[-1] được giả định là 0 nếu ta xét mảng 1-dựa).

3. Ứng dụng quan trọng nhất: Tính Tổng đoạn con trong O(1)

- Mục đích chính của Prefix Sum là trả lời các truy vấn tính tổng đoạn con (range sum queries) một cách nhanh chóng.
- Để tính tổng đoạn con từ chỉ mục **i** đến **j (i ≤ j):**

Sum(A[i … j]) = P[j] - P[i-1]

(Lưu ý: Nếu sử dụng mảng 0-dựa, ta cần xử lý trường hợp i=0 bằng cách coi P[-1] = 0).

IV. Thực Hành 1: Xây dựng và Tính tổng đoạn con (1D) (30 phút)

Ta sẽ cài đặt quá trình xây dựng mảng Prefix Sum và hàm tính tổng đoạn con.

1. Bài 1: Cài đặt Prefix Sum 1D

1.1. Code Python:

```python
class PrefixSum1D:
    # Cấu trúc dữ liệu Mảng cộng dồn 1 chiều (Prefix Sum Array).
    # Cho phép tính tổng đoạn con (range sum query) trong O(1).
    def __init__(self, arr):
        n = len(arr)
        # 1. Khởi tạo Mảng cộng dồn P (kích thước N + 1)
        # P[0] luôn là 0 để đơn giản hóa công thức tính tổng đoạn con từ chỉ mục 0.
        self.P = [0] * (n + 1)
        # 2. Xây dựng Mảng cộng dồn (Độ phức tạp O(N))
        # P[i+1] = Tổng của A[0] đến A[i]
        for i in range(n):
            # Công thức đệ quy: P[i+1] = P[i] + A[i]
            self.P[i + 1] = self.P[i] + arr[i]
        print(f"Mang goc A: {arr}")
        print(f"Prefix Sum P (kich thuoc {n+1}): {self.P}")
```

```
    def get_range_sum(self, i, j):
        # Tính tổng đoạn con A[i] đến A[j] (bao gồm i và j).
        # Tham số:
        #   i (int): Chỉ mục bắt đầu (0-dựa)
        #   j (int): Chỉ mục kết thúc (0-dựa)
        # Trả về:
        # Tổng của đoạn con A[i...j]. Độ phức tạp O(1).
        # Kiểm tra điều kiện chỉ mục hợp lệ
        if i > j or i < 0 or j >= len(self.P) - 1:
            raise ValueError("Chi muc khong hop le hoac nam ngoai pham vi mang
goc.")
        # Công thức chính: Sum(A[i...j]) = P[j+1] - P[i]
        # P[j+1] là Tổng từ A[0] đến A[j]
        # P[i] là Tổng từ A[0] đến A[i-1]
        # Hiệu số loại bỏ phần không cần thiết (A[0] đến A[i-1])
        return self.P[j + 1] - self.P[i]
# ---------------- CHẠY THỬ VÀ MINH HỌA O(1) ----------------
arr = [10, 20, 30, 40, 50]
ps = PrefixSum1D(arr)
# 1. Tính tổng A[1] đến A[3] (tức là 20 + 30 + 40 = 90)
# Công thức: P[3+1] - P[1] = P[4] - P[1] = 100 - 10 = 90
result = ps.get_range_sum(1, 3)
print(f"\nTinh tong A[1..3] (20+30+40): {result}")
# 2. Tính tổng A[0] đến A[2] (tức là 10 + 20 + 30 = 60)
# Công thức: P[2+1] - P[0] = P[3] - P[0] = 60 - 0 = 60
result_0 = ps.get_range_sum(0, 2)
print(f"Tinh tong A[0..2] (10+20+30): {result_0}")
```

=> Kết quả sau khi chạy chương trình:

```
Mang goc A: [10, 20, 30, 40, 50]
Prefix Sum P (kich thuoc 6): [0, 10, 30, 60, 100, 150]

Tinh tong A[1..3] (20+30+40): 90
Tinh tong A[0..2] (10+20+30): 60
```

- Giải thích Code Python:

+ Chỉ mục N+1: Việc tạo **self.P** có kích thước **N + 1** và đặt **P[0] = 0** giúp đơn giản hóa công thức.

+ Khi tính tổng từ **A[i]**: **P[i]** tương ứng với tổng *trước* **A[i]** (**A[0] đến A[i-1]**), do đó công thức trở thành **P[j+1] - P[i]**.

1.2. Code C++:

```
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
#include <stdexcept>         // Xử lí lỗi ngoại lệ
using namespace std;         // Tiền tố std
class PrefixSum1D
{
    // Cau truc du lieu Mang cong don 1 chieu (Prefix Sum Array).
    // Cho phep tinh tong doan con (range sum query) trong O(1).
private:
    vector<int> P;
public:
    PrefixSum1D(const vector<int>& arr)
    {
        int n = arr.size();
        // 1. Khoi tao Mang cong don P (kich thuoc N + 1)
```

```
        // P[0] luon la 0 de don gian hoa cong thuc tinh tong doan con tu chi muc
0.
        P.assign(n + 1, 0);
        // 2. Xay dung Mang cong don (Do phuc tap O(N))
        // P[i+1] = Tong cua A[0] den A[i]
        for (int i = 0; i < n; i++)
        {
            // Cong thuc de quy: P[i+1] = P[i] + A[i]
            P[i + 1] = P[i] + arr[i];
        }
        cout << "Mang goc A: ";
        cout << "[";
        for (int i = 0; i < n; i++)
        {
            cout << arr[i];
            if (i < n - 1) cout << ", ";
        }
        cout << "]" << endl;
        cout << "Prefix Sum P (kich thuoc " << n + 1 << "): ";
        cout << "[";
        for (int i = 0; i < P.size(); i++)
        {
            cout << P[i];
            if (i < P.size() - 1) cout << ", ";
        }
        cout << "]" << endl;
    }
    int get_range_sum(int i, int j)
    {
        // Tinh tong doan con A[i] den A[j] (bao gom i va j).
        // Tham so:
        //   i (int): Chi muc bat dau (0-dua)
        //   j (int): Chi muc ket thuc (0-dua)
        // Tra ve:
        // Tong cua doan con A[i...j]. Do phuc tap O(1).
        // Kiem tra dieu kien chi muc hop le
        if (i > j || i < 0 || j >= P.size() - 1)
        {
            throw invalid_argument("Chi muc khong hop le hoac nam ngoai pham vi
mang goc.");
        }
        // Cong thuc chinh: Sum(A[i...j]) = P[j+1] - P[i]
        // P[j+1] la Tong tu A[0] den A[j]
        // P[i] la Tong tu A[0] den A[i-1]
        // Hieu so loai bo phan khong can thiet (A[0] den A[i-1])
        return P[j + 1] - P[i];
    }
};
// ----------------- CHAY THU VA MINH HOA O(1) -----------------
int main()
{
    vector<int> arr = {10, 20, 30, 40, 50};
    PrefixSum1D ps(arr);
    // 1. Tinh tong A[1] den A[3] (tuc la 20 + 30 + 40 = 90)
    // Cong thuc: P[3+1] - P[1] = P[4] - P[1] = 100 - 10 = 90
    int result = ps.get_range_sum(1, 3);
    cout << "\nTinh tong A[1..3] (20+30+40): " << result << endl;
    // 2. Tinh tong A[0] den A[2] (tuc la 10 + 20 + 30 = 60)
```

```
    // Cong thuc: P[2+1] - P[0] = P[3] - P[0] = 60 - 0 = 60
    int result_0 = ps.get_range_sum(0, 2);
    cout << "Tinh tong A[0..2] (10+20+30): " << result_0 << endl;
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
Mang goc A: [10, 20, 30, 40, 50]
Prefix Sum P (kich thuoc 6): [0, 10, 30, 60, 100, 150]

Tinh tong A[1..3] (20+30+40): 90
Tinh tong A[0..2] (10+20+30): 60
```

V. Thực Hành 2: Prefix Sum nâng cao (25 phút)

Prefix Sum không chỉ dùng để tính tổng đoạn con mà còn giúp giải các bài toán yêu cầu so sánh tổng các phần tử trong mảng.

1. Bài 2: Tìm Chỉ mục Cân bằng (Equilibrium Index)

- **Định nghĩa:** Chỉ mục **i** được gọi là chỉ mục cân bằng nếu tổng các phần tử bên trái chỉ mục đó **bằng** tổng các phần tử bên phải chỉ mục đó.

+ **Tổng(A[0 … i-1]) = Tổng(A[i+1 … N-1])**

+ Ta có thể giải bài toán này trong O(N) thay vì $O(N^2)$ (cần tính tổng lại trong mỗi bước).

- **Sử dụng Prefix Sum:**

+ Tính P[i] (Tổng bên trái + A[i])

+ Tổng bên trái (LHS) của chỉ mục i là: P[i]

+ Tổng bên phải (RHS) của chỉ mục i là: Tổng toàn bộ - P[i+1]

+ Ta sẽ đơn giản hóa bằng cách duyệt và tính toán tổng bên trái và tổng bên phải tại chỗ.

1.1. Code Python:

```
def find_equilibrium_index(arr):
    # Tìm chỉ mục cân bằng (Equilibrium Index) của mảng.
    # Chỉ mục i là cân bằng nếu tổng các phần tử bên trái bằng tổng các phần tử bên
phải.
    # Độ phức tạp: O(N)
    # Tham số:
    # arr (list): Mảng số nguyên.
    # Trả về:
    # int: Chỉ mục cân bằng đầu tiên tìm thấy, hoặc -1 nếu không tồn tại.
    n = len(arr)
    if n == 0:
        return -1
    # BƯỚC 1: Tính tổng toàn bộ mảng (O(N))
    total_sum = sum(arr)
    # Khởi tạo tổng bên trái (Tổng của A[0...i-1])
    # Ban đầu, khi i=0, không có phần tử bên trái, nên left_sum = 0
    left_sum = 0
    # BƯỚC 2: Duyệt qua từng chỉ mục i (O(N))
    for i in range(n):
        # Tại chỉ mục i, ta có:
        # Tổng bên phải (Right Hand Side - RHS) = Tổng toàn bộ - Tổng bên trái -
arr[i]
        # 1. Tính tổng bên phải (RHS)
        right_sum = total_sum - left_sum - arr[i]
        # 2. Kiểm tra điều kiện cân bằng
```

```
        if left_sum == right_sum:
            # Tìm thấy chỉ mục cân bằng đầu tiên
            return i
        # 3. Cập nhật tổng bên trái cho lần lặp tiếp theo
        # left_sum sẽ trở thành tổng từ A[0] đến A[i]
        left_sum += arr[i]
    # Không tìm thấy chỉ mục cân bằng sau khi duyệt hết mảng
    return -1
# ---------------- CHẠY THỬ VÀ MINH HỌA O(N) ----------------
arr1 = [1, 2, 3, 4, 3, 2, 1]
# Chỉ mục 3 (giá trị 4): Tổng trái = 1+2+3 = 6. Tổng phải = 3+2+1 = 6.
print(f"Mang: {arr1} -> Chi muc can bang: {find_equilibrium_index(arr1)}")
arr2 = [1, 1, 1, 1, 1]
# Không có chỉ mục nào thỏa mãn.
print(f"Mang: {arr2} -> Chi muc can bang: {find_equilibrium_index(arr2)}")
arr3 = [1, 1, 2, 1, 1]
# Chỉ mục 2 (giá trị 2): Tổng trái = 1+1 = 2. Tổng phải = 1+1 = 2.
print(f"Mang: {arr3} -> Chi muc can bang: {find_equilibrium_index(arr3)}")
arr4 = [2, 0, -2, 4]
# Chỉ mục 3 (giá trị 4): Tổng trái = 2+0+(-2) = 0. Tổng phải = 0.
print(f"Mang: {arr4} -> Chi muc can bang: {find_equilibrium_index(arr4)}")
```

=> Kết quả sau khi chạy chương trình:

```
Mang: [1, 2, 3, 4, 3, 2, 1] -> Chi muc can bang: 3
Mang: [1, 1, 1, 1, 1] -> Chi muc can bang: 2
Mang: [1, 1, 2, 1, 1] -> Chi muc can bang: 2
Mang: [2, 0, -2, 4] -> Chi muc can bang: 1
```

- Giải thích Code Python:

Tối ưu hóa O(N): Ta chỉ cần một lần quét để tính **total_sum** và một lần quét nữa để kiểm tra từng chỉ mục **i**. Tại mỗi bước, ta tính **right_sum** dựa trên **total_sum** và **left_sum** đã tích lũy, tránh phải tính tổng lại nhiều lần.

1.2. Code C++:

```
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
using namespace std;         // Tiền tố std
int find_equilibrium_index(const vector<int>& arr)
{
    // Tim chi muc can bang (Equilibrium Index) cua mang.
    // Chi muc i la can bang neu tong cac phan tu ben trai bang tong cac phan tu
ben phai.
    // Do phuc tap: O(N)
    // Tham so:
    // arr (vector<int>): Mang so nguyen.
    // Tra ve:
    // int: Chi muc can bang dau tien tim thay, hoac -1 neu khong ton tai.
    int n = arr.size();
    if (n == 0)
    {
        return -1;
    }
    // BUOC 1: Tinh tong toan bo mang (O(N))
    int total_sum = 0;
    for (int x : arr)
    {
        total_sum += x;
    }
    // Khoi tao tong ben trai (Tong cua A[0...i-1])
```

```
    // Ban dau, khi i = 0, khong co phan tu ben trai, nen left_sum = 0
    int left_sum = 0;
    // BUOC 2: Duyet qua tung chi muc i (O(N))
    for (int i = 0; i < n; i++)
    {
        // Tai chi muc i, ta co:
        // Tong ben phai (Right Hand Side - RHS) = Tong toan bo - Tong ben trai -
arr[i]
        // 1. Tinh tong ben phai (RHS)
        int right_sum = total_sum - left_sum - arr[i];
        // 2. Kiem tra dieu kien can bang
        if (left_sum == right_sum)
        {
            // Tim thay chi muc can bang dau tien
            return i;
        }
        // 3. Cap nhat tong ben trai cho lan lap tiep theo
        // left_sum se tro thanh tong tu A[0] den A[i]
        left_sum += arr[i];
    }
    // Khong tim thay chi muc can bang sau khi duyet het mang
    return -1;
}
// ----------------- CHAY THU VA MINH HOA O(N) -----------------
int main()
{
    vector<int> arr1 = {1, 2, 3, 4, 3, 2, 1};
    // Chi muc 3 (gia tri 4): Tong trai = 1+2+3 = 6. Tong phai = 3+2+1 = 6.
    cout << "Mang: ";
    for (int x : arr1) cout << x << " ";
    cout << "-> Chi muc can bang: " << find_equilibrium_index(arr1) << endl;
    vector<int> arr2 = {1, 1, 1, 1, 1};
    // Khong co chi muc nao thoa man.
    cout << "Mang: ";
    for (int x : arr2) cout << x << " ";
    cout << "-> Chi muc can bang: " << find_equilibrium_index(arr2) << endl;
    vector<int> arr3 = {1, 1, 2, 1, 1};
    // Chi muc 2 (gia tri 2): Tong trai = 1+1 = 2. Tong phai = 1+1 = 2.
    cout << "Mang: ";
    for (int x : arr3) cout << x << " ";
    cout << "-> Chi muc can bang: " << find_equilibrium_index(arr3) << endl;
    vector<int> arr4 = {2, 0, -2, 4};
    // Chi muc 3 (gia tri 4): Tong trai = 2+0+(-2) = 0. Tong phai = 0.
    cout << "Mang: ";
    for (int x : arr4) cout << x << " ";
    cout << "-> Chi muc can bang: " << find_equilibrium_index(arr4) << endl;
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
Mang: 1 2 3 4 3 2 1 -> Chi muc can bang: 3
Mang: 1 1 1 1 1 -> Chi muc can bang: 2
Mang: 1 1 2 1 1 -> Chi muc can bang: 2
Mang: 2 0 -2 4 -> Chi muc can bang: 1
```

- Giải thích Code C++:

Kiểu dữ liệu: Khi viết chương trình nên sử dụng **long long** cho các tổng là cần thiết để tránh lỗi tràn số **nếu mảng có kích thước lớn**.

VI. Tổng kết & Bài tập Về nhà (10 phút)

1. Tóm tắt:

Mảng cộng dồn là một kỹ thuật mạnh mẽ giúp giảm độ phức tạp của các truy vấn tính tổng đoạn con từ O(N) xuống O(1), nhưng phải trả chi phí O(N) ban đầu để xây dựng mảng.

Phương pháp	**Xây dựng**	**Truy vấn (Query)**
Vòng lặp thuần	O(1)	O(N)
Prefix Sum	O(N)	O(1)

2. Bài tập Về nhà:

2.1. Bài tập 1 (Prefix Sum 2D): Nghiên cứu và viết code xây dựng **Prefix Sum Matrix (Mảng cộng dồn 2D)** để tính tổng của một vùng hình chữ nhật bất kỳ trong ma trận trong O(1).

2.1.1. Code Python:

```
class PrefixSum2D:
    # Cấu trúc dữ liệu Mảng cộng dồn 2 chiều (Prefix Sum Matrix).
    # Cho phép tính tổng vùng hình chữ nhật bất kỳ trong O(1).
    def __init__(self, matrix):
        R = len(matrix)
        if R == 0:
            self.P = [[]]
            return
        C = len(matrix[0])
        # Kích thước P: (R + 1) x (C + 1) để xử lý biên dễ dàng (P[0][c] = 0,
P[r][0] = 0)
        self.P = [[0] * (C + 1) for _ in range(R + 1)]
        # Xây dựng Prefix Sum Matrix (O(R*C))
        for r in range(R):
            for c in range(C):
                # r_idx = r + 1, c_idx = c + 1 (Sử dụng chỉ mục 1-dựa cho mảng P)
                r_idx = r + 1
                c_idx = c + 1
                # Công thức: P[r,c] = A[r-1, c-1] + P[r-1, c] + P[r, c-1] - P[r-1,
c-1]
                self.P[r_idx][c_idx] = (
                    matrix[r][c]
                    + self.P[r_idx - 1][c_idx]      # Vùng trên
                    + self.P[r_idx][c_idx - 1]      # Vùng trái
                    - self.P[r_idx - 1][c_idx - 1]  # Trừ đi vùng giao thoa
                )
        print("Ma tran Goc:")
        for row in matrix:
            print(row)
        print("\nPrefix Sum Matrix P:")
        for row in self.P:
            print(row)
    def get_range_sum(self, r1, c1, r2, c2):
        # Tính tổng vùng hình chữ nhật từ (r1, c1) đến (r2, c2) trong O(1).
        # r1, c1 là góc trên-trái; r2, c2 là góc dưới-phải (chỉ mục 0-dựa).
        # Tăng chỉ mục để phù hợp với mảng P (1-dựa)
        r2_idx = r2 + 1
        c2_idx = c2 + 1
        r1_idx_minus_1 = r1  # Tương ứng với r1 - 1
```

```
        c1_idx_minus_1 = c1  # Tương ứng với c1 - 1
        # Ứng dụng công thức Sum = P[r2, c2] - P[r1-1, c2] - P[r2, c1-1] + P[r1-1,
c1-1]
        total = self.P[r2_idx][c2_idx]
        # Trừ vùng trên
        top_area = self.P[r1_idx_minus_1][c2_idx]
        # Trừ vùng trái
        left_area = self.P[r2_idx][c1_idx_minus_1]
        # Cộng bù vùng giao thoa
        overlap_area = self.P[r1_idx_minus_1][c1_idx_minus_1]
        return total - top_area - left_area + overlap_area
# ----------------- CHẠY THỬ VÀ MINH HỌA O(1) -----------------
matrix = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]
ps2d = PrefixSum2D(matrix)
# 1. Tính tổng toàn bộ ma trận (0, 0) đến (2, 2)
r1, c1 = 0, 0
r2, c2 = 2, 2
# Kết quả mong đợi: 1+2+...+9 = 45
result1 = ps2d.get_range_sum(r1, c1, r2, c2)
print(f"\nTong vung ({r1},{c1}) den ({r2},{c2}): {result1} (Mong doi: 45)")
# 2. Tính tổng vùng con (1, 1) đến (2, 2)
r1, c1 = 1, 1
r2, c2 = 2, 2
# Kết quả mong đợi: 5 + 6 + 8 + 9 = 28
result2 = ps2d.get_range_sum(r1, c1, r2, c2)
print(f"Tong vung ({r1},{c1}) den ({r2},{c2}): {result2} (Mong doi: 28)")
# 3. Tính tổng một ô đơn lẻ (1, 0)
r1, c1 = 1, 0
r2, c2 = 1, 0
# Kết quả mong đợi: 4
result3 = ps2d.get_range_sum(r1, c1, r2, c2)
print(f"Tong vung ({r1},{c1}) den ({r2},{c2}): {result3} (Mong doi: 4)")
```

=> Kết quả sau khi chạy chương trình:

```
Ma tran Goc:
[1, 2, 3]
[4, 5, 6]
[7, 8, 9]

Prefix Sum Matrix P:
[0, 0, 0, 0]
[0, 1, 3, 6]
[0, 5, 12, 21]
[0, 12, 27, 45]

Tong vung (0,0) den (2,2): 45 (Mong doi: 45)
Tong vung (1,1) den (2,2): 28 (Mong doi: 28)
Tong vung (1,0) den (1,0): 4 (Mong doi: 4)
```

2.1.2. Code C++:

```
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
using namespace std;         // Tiền tố std
class PrefixSum2D
```

```
{
    // Cau truc du lieu Mang cong don 2 chieu (Prefix Sum Matrix).
    // Cho phep tinh tong vung hinh chu nhat bat ky trong O(1).
private:
    vector<vector<int>> P;
public:
    PrefixSum2D(const vector<vector<int>>& matrix)
    {
        int R = matrix.size();
        if (R == 0)
        {
            P = vector<vector<int>>(1, vector<int>());
            return;
        }
        int C = matrix[0].size();
        // Kich thuoc P: (R + 1) x (C + 1) de xu ly bien de dang
        // (P[0][c] = 0, P[r][0] = 0)
        P = vector<vector<int>>(R + 1, vector<int>(C + 1, 0));
        // Xay dung Prefix Sum Matrix (O(R*C))
        for (int r = 0; r < R; r++)
        {
            for (int c = 0; c < C; c++)
            {
                // r_idx = r + 1, c_idx = c + 1 (Su dung chi muc 1-dua cho mang P)
                int r_idx = r + 1;
                int c_idx = c + 1;
                // Cong thuc:
                // P[r,c] = A[r-1,c-1] + P[r-1,c] + P[r,c-1] - P[r-1,c-1]
                P[r_idx][c_idx] =
                    matrix[r][c]
                    + P[r_idx - 1][c_idx]      // Vung tren
                    + P[r_idx][c_idx - 1]      // Vung trai
                    - P[r_idx - 1][c_idx - 1]; // Tru vung giao thoa
            }
        }
        cout << "Ma tran Goc:" << endl;
        for (const auto& row : matrix)
        {
            for (int x : row) cout << x << " ";
            cout << endl;
        }
        cout << "\nPrefix Sum Matrix P:" << endl;
        for (const auto& row : P)
        {
            for (int x : row) cout << x << " ";
            cout << endl;
        }
    }
    int get_range_sum(int r1, int c1, int r2, int c2)
    {
        // Tinh tong vung hinh chu nhat tu (r1, c1) den (r2, c2) trong O(1).
        // r1, c1 la goc tren-trai; r2, c2 la goc duoi-phai (chi muc 0-dua).
        // Tang chi muc de phu hop voi mang P (1-dua)
        int r2_idx = r2 + 1;
        int c2_idx = c2 + 1;
        int r1_idx_minus_1 = r1;
        int c1_idx_minus_1 = c1;
        // Ung dung cong thuc:
```

```
        // Sum = P[r2,c2] - P[r1-1,c2] - P[r2,c1-1] + P[r1-1,c1-1]
        int total = P[r2_idx][c2_idx];
        // Tru vung tren
        int top_area = P[r1_idx_minus_1][c2_idx];
        // Tru vung trai
        int left_area = P[r2_idx][c1_idx_minus_1];
        // Cong bu vung giao thoa
        int overlap_area = P[r1_idx_minus_1][c1_idx_minus_1];
        return total - top_area - left_area + overlap_area;
    }
};
// ----------------- CHAY THU VA MINH HOA O(1) -----------------
int main()
{
    vector<vector<int>> matrix = {
        {1, 2, 3},
        {4, 5, 6},
        {7, 8, 9}
    };
    PrefixSum2D ps2d(matrix);
    // 1. Tinh tong toan bo ma tran (0,0) den (2,2)
    int r1 = 0, c1 = 0, r2 = 2, c2 = 2;
    // Ket qua mong doi: 45
    int result1 = ps2d.get_range_sum(r1, c1, r2, c2);
    cout << "\nTong vung (" << r1 << "," << c1 << ") den ("
         << r2 << "," << c2 << "): " << result1 << " (Mong doi: 45)" << endl;
    // 2. Tinh tong vung con (1,1) den (2,2)
    r1 = 1; c1 = 1; r2 = 2; c2 = 2;
    // Ket qua mong doi: 28
    int result2 = ps2d.get_range_sum(r1, c1, r2, c2);
    cout << "Tong vung (" << r1 << "," << c1 << ") den ("
         << r2 << "," << c2 << "): " << result2 << " (Mong doi: 28)" << endl;
    // 3. Tinh tong mot o don le (1,0)
    r1 = 1; c1 = 0; r2 = 1; c2 = 0;
    // Ket qua mong doi: 4
    int result3 = ps2d.get_range_sum(r1, c1, r2, c2);
    cout << "Tong vung (" << r1 << "," << c1 << ") den ("
         << r2 << "," << c2 << "): " << result3 << " (Mong doi: 4)" << endl;
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
Ma tran Goc:
1 2 3
4 5 6
7 8 9

Prefix Sum Matrix P:
0 0 0 0
0 1 3 6
0 5 12 21
0 12 27 45


Tong vung (0,0) den (2,2): 45 (Mong doi: 45)
Tong vung (1,1) den (2,2): 28 (Mong doi: 28)
Tong vung (1,0) den (1,0): 4 (Mong doi: 4)
```

2.2. Bài tập 2 (Tìm đoạn con): Cho một mảng số nguyên và một số **K**. Sử dụng **Prefix Sum**, tìm xem có tồn tại một đoạn con (Subarray) nào có tổng bằng **K** hay không. (Gợi ý: Sử dụng **Hash Map** để lưu trữ các **prefix sum** đã thấy).

2.2.1. Code Python:

```
def find_subarray_with_sum_k(arr, K):
    # Sử dụng Prefix Sum và Hash Map để kiểm tra xem có tồn tại
    # một đoạn con (subarray) có tổng bằng K hay không.
    # Độ phức tạp thời gian: O(N)
    # Độ phức tạp không gian: O(N) (cho Hash Map)
    # Tham số:
    # arr (list): Mảng số nguyên.
    # K (int): Giá trị tổng mục tiêu.
    # Trả về:
    # bool: True nếu tìm thấy đoạn con, False nếu không.
    # 1. Khởi tạo Hash Map (Dictionary)
    # Lưu trữ: { prefix_sum: count_of_occurrences }
    # Bắt đầu với prefix_sum = 0 đã xuất hiện 1 lần (để xử lý trường hợp tổng từ
đầu mảng)
    prefix_sums = {0: 1}
    current_sum = 0
    # 2. Duyệt qua mảng và tính Prefix Sum (O(N))
    for num in arr:
        current_sum += num
        # Giá trị cần tìm: prefix_sum[i-1] = prefix_sum[j] - K
        target_prefix_sum = current_sum - K
        # 3. Kiểm tra trong Hash Map
        # Nếu target_prefix_sum đã từng xuất hiện, tức là đã tìm thấy đoạn con có
tổng bằng K
        if target_prefix_sum in prefix_sums:
            return True
        # 4. Cập nhật Hash Map với prefix sum hiện tại
        # (Sử dụng .get(key, 0) + 1 để tăng count hoặc khởi tạo là 1)
        prefix_sums[current_sum] = prefix_sums.get(current_sum, 0) + 1
    return False
# ---------------- CHẠY THỬ VÀ MINH HỌA ----------------
arr1 = [10, 5, 2, 7, 1, 9]
K1 = 15
# Đoạn con [10, 5] hoặc [5, 2, 7, 1] (5+2+7+1 = 15)
print(f"Mang: {arr1}, K={K1} -> Co ton tai doan con tong 15:
{find_subarray_with_sum_k(arr1, K1)}")
arr2 = [1, 2, 3, 4, 5]
K2 = 10
# Đoạn con [1, 2, 3, 4]
print(f"Mang: {arr2}, K={K2} -> Co ton tai doan con tong 10:
{find_subarray_with_sum_k(arr2, K2)}")
arr3 = [1, 2, 3, 4, 5]
K3 = 100
# Không tồn tại
print(f"Mang: {arr3}, K={K3} -> Co ton tai doan con tong 100:
{find_subarray_with_sum_k(arr3, K3)}")
arr4 = [3, -2, 4, -1, 5]
K4 = 7
# Đoạn con [3, -2, 4, -1, 5] (Tổng = 9) hoặc [3, -2, 4, -1] (Tổng = 4)
# Đoạn con [3, -2, 4, -1, 5]
# Đoạn con [4, -1, 5] (Tổng = 8)
# Đoạn con [3, -2, 4, -1, 5]
```

```
# Đoạn con [4, -1, 5]
# Đoạn con [3, -2, 4, -1] (Tổng = 4)
# Đoạn con [3, -2, 4, -1, 5] (Tổng = 9)
# Đoạn con [3, -2, 4, -1, 5] (Tổng = 9)
# Đoạn con [3, -2, 4, -1, 5] (Tổng = 9)
# Đoạn con [3, -2, 4, -1, 5] (Tổng = 9)
# Đoạn con [3, -2, 4, -1, 5] (Tổng = 9)
# Đoạn con [3, -2, 4, -1, 5] (Tổng = 9)
# Đoạn con [4, 3] = 7
print(f"Mang: {arr4}, K={K4} -> Co ton tai doan con tong 7:
{find_subarray_with_sum_k(arr4, K4)}")
```

=> Kết quả sau khi chạy chương trình:

```
Mang: [10, 5, 2, 7, 1, 9], K=15 -> Co ton tai doan con tong 15: True
Mang: [1, 2, 3, 4, 5], K=10 -> Co ton tai doan con tong 10: True
Mang: [1, 2, 3, 4, 5], K=100 -> Co ton tai doan con tong 100: False
Mang: [3, -2, 4, -1, 5], K=7 -> Co ton tai doan con tong 7: False
```

2.2.2. Code C++:

```
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
using namespace std;         // Tiền tố std
bool find_subarray_with_sum_k(const vector<int>& arr, int K)
{
    // Su dung Prefix Sum va Hash Map de kiem tra xem co ton tai
    // mot doan con (subarray) co tong bang K hay khong.
    // Do phuc tap thoi gian: O(N)
    // Do phuc tap khong gian: O(N) (cho Hash Map)
    // Tham so:
    // arr (vector<int>): Mang so nguyen.
    // K (int): Gia tri tong muc tieu.
    // Tra ve:
    // bool: True neu tim thay doan con, False neu khong.
    // 1. Khoi tao Hash Map
    // Luu tru: { prefix_sum: count_of_occurrences }
    // Bat dau voi prefix_sum = 0 da xuat hien 1 lan
    unordered_map<int, int> prefix_sums;
    prefix_sums[0] = 1;
    int current_sum = 0;
    // 2. Duyet qua mang va tinh Prefix Sum (O(N))
    for (int num : arr)
    {
        current_sum += num;
        // Gia tri can tim: prefix_sum[i-1] = prefix_sum[j] - K
        int target_prefix_sum = current_sum - K;
        // 3. Kiem tra trong Hash Map
        // Neu target_prefix_sum da tung xuat hien
        // tuc la da tim thay doan con co tong bang K
        if (prefix_sums.find(target_prefix_sum) != prefix_sums.end())
        {
            return true;
        }
        // 4. Cap nhat Hash Map voi prefix sum hien tai
        prefix_sums[current_sum]++;
    }
    return false;
}
// ---------------- CHAY THU VA MINH HOA ----------------
int main()
```

```
{
    vector<int> arr1 = {10, 5, 2, 7, 1, 9};
    int K1 = 15;
    cout << "Mang: ";
    for (int x : arr1) cout << x << " ";
    cout << ", K=" << K1
         << " -> Co ton tai doan con tong 15: "
         << find_subarray_with_sum_k(arr1, K1) << endl;
    vector<int> arr2 = {1, 2, 3, 4, 5};
    int K2 = 10;
    cout << "Mang: ";
    for (int x : arr2) cout << x << " ";
    cout << ", K=" << K2
         << " -> Co ton tai doan con tong 10: "
         << find_subarray_with_sum_k(arr2, K2) << endl;
    vector<int> arr3 = {1, 2, 3, 4, 5};
    int K3 = 100;
    cout << "Mang: ";
    for (int x : arr3) cout << x << " ";
    cout << ", K=" << K3
         << " -> Co ton tai doan con tong 100: "
         << find_subarray_with_sum_k(arr3, K3) << endl;
    vector<int> arr4 = {3, -2, 4, -1, 5};
    int K4 = 7;
    cout << "Mang: ";
    for (int x : arr4) cout << x << " ";
    cout << ", K=" << K4
         << " -> Co ton tai doan con tong 7: "
         << find_subarray_with_sum_k(arr4, K4) << endl;
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
Mang: 10 5 2 7 1 9 , K=15 -> Co ton tai doan con tong 15: 1
Mang: 1 2 3 4 5 , K=10 -> Co ton tai doan con tong 10: 1
Mang: 1 2 3 4 5 , K=100 -> Co ton tai doan con tong 100: 0
Mang: 3 -2 4 -1 5 , K=7 -> Co ton tai doan con tong 7: 0
```

2.3. Bài tập 3 (Max Subarray Sum): Giải quyết bài toán tìm tổng lớn nhất của một đoạn con trong mảng (Max Subarray Sum) bằng cách sử dụng Prefix Sum (tính toán P[j] - P[i]). So sánh với thuật toán **Kadane**.

2.3.1. Code Python:

```
def max_subarray_sum_prefix_sum(arr):
    # Tìm tổng lớn nhất của đoạn con sử dụng kỹ thuật Prefix Sum.
    # Độ phức tạp: O(N) thời gian, O(N) không gian.
    n = len(arr)
    if n == 0:
        return 0
    # 1. Xây dựng Prefix Sum Array P (kích thước N+1)
    # P[0] = 0
    P = [0] * (n + 1)
    for i in range(n):
        P[i + 1] = P[i] + arr[i]
    # 2. Duyệt qua mảng Prefix Sum để tìm Max Sum
    max_so_far = arr[0] # Khởi tạo với phần tử đầu tiên (hoặc âm vô cùng)
    min_prefix_sum = 0 # Luôn khởi tạo là P[0] = 0 (Tổng rỗng)
    # Bắt đầu duyệt từ P[1] (tổng của A[0])
    # i đại diện cho chỉ mục kết thúc (j) trong công thức P[j+1]
```

```
    for i in range(1, n + 1):
        # P[i] là Prefix Sum hiện tại (tương ứng với PrefixSum[j])
        current_prefix_sum = P[i]
        # Tính tổng đoạn con lớn nhất kết thúc tại chỉ mục i-1 của mảng gốc
        current_max_sum = current_prefix_sum - min_prefix_sum
        # Cập nhật tổng lớn nhất toàn cục
        if current_max_sum > max_so_far:
            max_so_far = current_max_sum
        # Cập nhật giá trị Prefix Sum nhỏ nhất đã thấy
        if current_prefix_sum < min_prefix_sum:
            min_prefix_sum = current_prefix_sum
    return max_so_far
# ----------------- CHẠY THỬ -----------------
arr = [-2, 1, -3, 4, -1, 2, 1, -5, 4]
result_ps = max_subarray_sum_prefix_sum(arr)
print(f"Mang: {arr}")
print(f"Max Subarray Sum (Prefix Sum): {result_ps}") # Kết quả mong đợi: 6 ([4, -1,
2, 1])
```

=> Kết quả sau khi chạy chương trình:

```
Mang: [-2, 1, -3, 4, -1, 2, 1, -5, 4]
Max Subarray Sum (Prefix Sum): 6
```

2.3.2. Code C++:

```
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
using namespace std;         // Tiền tố std
int max_subarray_sum_prefix_sum(const vector<int>& arr)
{
    // Tim tong lon nhat cua doan con su dung ky thuat Prefix Sum.
    // Do phuc tap: O(N) thoi gian, O(N) khong gian.
    int n = arr.size();
    if (n == 0)
    {
        return 0;
    }
    // 1. Xay dung Prefix Sum Array P (kich thuoc N+1)
    // P[0] = 0
    vector<int> P(n + 1, 0);
    for (int i = 0; i < n; i++)
    {
        P[i + 1] = P[i] + arr[i];
    }
    // 2. Duyet qua mang Prefix Sum de tim Max Sum
    int max_so_far = arr[0]; // Khoi tao voi phan tu dau tien
    int min_prefix_sum = 0;  // Luon khoi tao la P[0] = 0 (Tong rong)
    // Bat dau duyet tu P[1] (tong cua A[0])
    // i dai dien cho chi muc ket thuc (j) trong cong thuc P[j+1]
    for (int i = 1; i <= n; i++)
    {
        // P[i] la Prefix Sum hien tai (tuong ung voi PrefixSum[j])
        int current_prefix_sum = P[i];
        // Tinh tong doan con lon nhat ket thuc tai chi muc i-1 cua mang goc
        int current_max_sum = current_prefix_sum - min_prefix_sum;
        // Cap nhat tong lon nhat toan cuc
        if (current_max_sum > max_so_far)
        {
            max_so_far = current_max_sum;
        }
```

```
        // Cap nhat gia tri Prefix Sum nho nhat da thay
        if (current_prefix_sum < min_prefix_sum)
        {
            min_prefix_sum = current_prefix_sum;
        }
    }
    return max_so_far;
}
// ----------------- CHAY THU -----------------
int main()
{
    vector<int> arr = {-2, 1, -3, 4, -1, 2, 1, -5, 4};
    int result_ps = max_subarray_sum_prefix_sum(arr);
    cout << "Mang: ";
    for (int x : arr)
    {
        cout << x << " ";
    }
    cout << endl;
    cout << "Max Subarray Sum (Prefix Sum): " << result_ps << endl;
    // Ket qua mong doi: 6 ([4, -1, 2, 1])
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
Mang: -2 1 -3 4 -1 2 1 -5 4
Max Subarray Sum (Prefix Sum): 6
```

* ***So sánh với Thuật toán Kadane:***

- **Thuật toán Kadane:** Là phương pháp tối ưu và phổ biến nhất để giải quyết bài toán Max Subarray Sum.

- **Nguyên lý cốt lõi:** Thuật toán này duyệt qua mảng một lần và tại mỗi vị trí **i**, nó duy trì hai biến:

+ **max_ending_here:** Tổng lớn nhất của đoạn con kết thúc tại vị trí hiện tại **max_ending_here = max(A[i], max_ending_here + A[i])**

+ **max_so_far:** Tổng lớn nhất toàn cục tìm thấy cho đến nay.

2.3.3. Code Python (Thuật toán Kadane):

```
def max_subarray_sum_kadane(arr):
    # Tìm tổng lớn nhất của đoạn con sử dụng Thuật toán Kadane.
    # Độ phức tạp: O(N) thời gian, O(1) không gian.
    n = len(arr)
    if n == 0:
        return 0
    # Khởi tạo cả hai biến với phần tử đầu tiên
    max_so_far = arr[0]
    max_ending_here = arr[0]
    for i in range(1, n):
        # 1. Cập nhật max_ending_here:
        # Lớn hơn giữa: (phần tử hiện tại) VÀ (tổng cũ + phần tử hiện tại)
        max_ending_here = max(arr[i], max_ending_here + arr[i])
        # 2. Cập nhật max_so_far:
        # Lớn hơn giữa: (tổng lớn nhất đã biết) VÀ (tổng lớn nhất kết thúc tại đây)
        max_so_far = max(max_so_far, max_ending_here)
    return max_so_far
# ----------------- CHẠY THỬ -----------------
arr = [-2, 1, -3, 4, -1, 2, 1, -5, 4]
```

```
result_kadane = max_subarray_sum_kadane(arr)
print(f"Max Subarray Sum (Kadane's): {result_kadane}") # Kết quả: 6
```

=> Kết quả sau khi chạy chương trình:

```
Max Subarray Sum (Kadane's): 6
```

2.3.4. Code C++ (Thuật toán Kadane):

```
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
using namespace std;         // Tiền tố std
int max_subarray_sum_kadane(const vector<int>& arr)
{
    // Tim tong lon nhat cua doan con su dung Thuat toan Kadane.
    // Do phuc tap: O(N) thoi gian, O(1) khong gian.
    int n = arr.size();
    if (n == 0)
    {
        return 0;
    }
    // Khoi tao ca hai bien voi phan tu dau tien
    int max_so_far = arr[0];
    int max_ending_here = arr[0];
    for (int i = 1; i < n; i++)
    {
        // 1. Cap nhat max_ending_here:
        // Lon hon giua: (phan tu hien tai) VA (tong cu + phan tu hien tai)
        max_ending_here = max(arr[i], max_ending_here + arr[i]);
        // 2. Cap nhat max_so_far:
        // Lon hon giua: (tong lon nhat da biet) VA (tong lon nhat ket thuc tai
day)
        max_so_far = max(max_so_far, max_ending_here);
    }
    return max_so_far;
}
// ----------------- CHAY THU -----------------
int main()
{
    vector<int> arr = {-2, 1, -3, 4, -1, 2, 1, -5, 4};
    int result_kadane = max_subarray_sum_kadane(arr);
    cout << "Max Subarray Sum (Kadane's): " << result_kadane << endl;
    // Ket qua: 6
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
Max Subarray Sum (Kadane's): 6
```

* ***Bảng so sánh chi tiết giữa Prefix Sum và Kadane:***

Tiêu chí	Prefix Sum Approach	Kadane's Algorithm
Độ phức tạp Thời gian	O(N)	O(N)
Độ phức tạp Không gian	O(N) (Lưu mảng Prefix Sum)	O(1) (Không cần lưu trữ phụ)
Khái niệm	Phải tính toán và lưu trữ tổng của các đoạn con bắt đầu từ 0.	Chỉ cần duy trì trạng thái hiện tại (max_ending_here) và trạng thái tốt nhất (max_so_far).

Sự phổ biến/Tối ưu	Ít tối ưu hơn (do không gian O(N)). Dễ mở rộng cho các bài toán phức tạp hơn (như Max 2D Subarray Sum).	**Là phương pháp tối ưu nhất** cho bài toán 1D.

CHỦ ĐỀ 9. QUY HOẠCH ĐỘNG (DP) CƠ BẢN

BÀI 25. GIỚI THIỆU DP & TÌM DÃY CON LIÊN TIẾP CÓ TỔNG LỚN NHẤT (KADANE)

I. Mục Tiêu:

- **Kiến thức:** HS hiểu khái niệm Quy hoạch động (Dynamic Programming - DP) và các thành phần cốt lõi: **Bài toán con chồng chéo (Overlapping Subproblems)** và **Cấu trúc tối ưu tối ưu (Optimal Substructure)**. Nắm vững hai kỹ thuật DP dãy con: Kadane (Max Subarray Sum) và DP cơ bản (LIS).
- **Kỹ năng:** HS cài đặt được thuật toán Kadane và thuật toán LIS DP $O(N^2)$ trong Python và C++.
- **Tư duy:** HS nhận diện được khi nào một bài toán dãy con có thể được giải quyết bằng DP.

II. Phân Bổ Thời Gian (90 Phút):

Hoạt Động	Nội Dung	Thời gian (Phút)
Ôn tập & Giới thiệu	Định nghĩa DP, 2 điều kiện, so sánh với Recursion	15
Thực hành 1	Bài 1. Max Subarray Sum (Thuật toán Kadane - DP tuyến tính)	30
Thực hành 2	Bài 2. Longest Increasing Subsequence (LIS - DP $O(N^2)$)	35
Tổng kết	So sánh hai bài toán, Bài tập về nhà	10
Tổng cộng		**90**

III. Lý Thuyết: Giới thiệu Quy hoạch động (15 phút):

1. Quy hoạch động (Dynamic Programming - DP) là gì?

DP là một phương pháp tối ưu hóa được sử dụng để giải quyết các bài toán phức tạp bằng cách chia chúng thành các **bài toán con đơn giản hơn** và lưu trữ kết quả của các bài toán con đó để tránh tính toán lại.

2. Hai điều kiện cốt lõi của DP:

- **Bài toán con chồng chéo (Overlapping Subproblems):** Cùng một bài toán con được giải quyết nhiều lần. DP lưu trữ kết quả (thường trong mảng DP) để tái sử dụng.

- **Cấu trúc tối ưu tối ưu (Optimal Substructure):** Lời giải tối ưu của bài toán lớn có thể được xây dựng từ lời giải tối ưu của các bài toán con.

3. DP Dãy con (Subsequence DP):

Các bài toán liên quan đến việc tìm kiếm các đặc tính tối ưu trong một dãy con (Subarray, Subsequence) thường tuân theo quy tắc DP, nơi mà lời giải tại vị trí **i** phụ thuộc vào lời giải tối ưu đã được tính toán ở các vị trí **j < i**.

IV. Thực Hành 1: Max Subarray Sum (Kadane's Algorithm) (30 phút)

1. Bài 1: Tìm tổng lớn nhất của một đoạn con liên tục trong mảng.

- **Định nghĩa DP:** Mặc dù **Kadane** không cần mảng **DP** tường minh, nó là một dạng Quy hoạch động Tuyến tính.
- **Trạng thái DP: DP[i]** (hoặc biến **max_ending_here**) là tổng lớn nhất của đoạn con **kết thúc tại chỉ mục i**.
- **Công thức Truy hồi:** Tổng lớn nhất kết thúc tại **i** là:

DP[i] = max(A[i], DP[i-1] + A[i])

1.1. Code Python:

```python
def max_subarray_sum_kadane(arr):
    # Tìm tổng lớn nhất của đoạn con liên tục bằng Thuật toán Kadane.
    # O(N) thời gian, O(1) không gian.
    if not arr:
        return 0
    # Khởi tạo: Bắt đầu bằng phần tử đầu tiên
    max_so_far = arr[0]      # Tổng lớn nhất toàn cục tìm thấy cho đến nay
    max_ending_here = arr[0] # Tổng lớn nhất của đoạn con kết thúc tại vị trí hiện
tại
    for i in range(1, len(arr)):
        current_val = arr[i]
        # 1. Áp dụng công thức DP: Quyết định mở rộng đoạn cũ hay bắt đầu đoạn mới
        # Nếu max_ending_here + current_val lớn hơn current_val, tiếp tục mở rộng
đoạn cũ
        # Ngược lại, bắt đầu đoạn mới từ current_val
        max_ending_here = max(current_val, max_ending_here + current_val)
        # 2. Cập nhật tổng lớn nhất toàn cục
        max_so_far = max(max_so_far, max_ending_here)
    return max_so_far
# Chạy thử
arr1 = [-2, 1, -3, 4, -1, 2, 1, -5, 4]
# Đoạn con lớn nhất là [4, -1, 2, 1] có tổng là 6
print(f"Mang: {arr1}")
print(f"Max Subarray Sum: {max_subarray_sum_kadane(arr1)}")
```

=> Kết quả sau khi chạy chương trình:

```
Mang: [-2, 1, -3, 4, -1, 2, 1, -5, 4]
Max Subarray Sum: 6
```

- Giải thích Code Python:

Nguyên tắc: Nếu DP[i-1] dương, ta nên mở rộng đoạn con cũ. Nếu DP[i-1] âm, tốt hơn là bắt đầu đoạn con mới từ A[i]. Kadane đạt O(N) thời gian và O(1) không gian.

1.2. Code C++:

```cpp
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
using namespace std;         // Tiền tố std
int max_subarray_sum_kadane(const vector<int>& arr)
```

```
{
    // Tim tong lon nhat cua doan con lien tuc bang Thuat toan Kadane.
    // O(N) thoi gian, O(1) khong gian.
    if (arr.empty())
    {
        return 0;
    }
    // Khoi tao: Bat dau bang phan tu dau tien
    int max_so_far = arr[0];       // Tong lon nhat toan cuc tim thay cho den nay
    int max_ending_here = arr[0];  // Tong lon nhat cua doan con ket thuc tai vi
tri hien tai
    for (int i = 1; i < arr.size(); i++)
    {
        int current_val = arr[i];
        // 1. Ap dung cong thuc DP: Quyet dinh mo rong doan cu hay bat dau doan moi
        // Neu max_ending_here + current_val lon hon current_val, tiep tuc mo rong
doan cu
        // Nguoc lai, bat dau doan moi tu current_val
        max_ending_here = max(current_val, max_ending_here + current_val);
        // 2. Cap nhat tong lon nhat toan cuc
        max_so_far = max(max_so_far, max_ending_here);
    }
    return max_so_far;
}
// Chay thu
int main()
{
    vector<int> arr1 = {-2, 1, -3, 4, -1, 2, 1, -5, 4};
    // Doan con lon nhat la [4, -1, 2, 1] co tong la 6
    cout << "Mang: ";
    for (int x : arr1)
    {
        cout << x << " ";
    }
    cout << endl;
    cout << "Max Subarray Sum: " << max_subarray_sum_kadane(arr1) << endl;
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
Mang: -2 1 -3 4 -1 2 1 -5 4
Max Subarray Sum: 6
```

- **Giải thích Code C++:** Logic hoạt động giống như Python, tận dụng thuộc tính DP để đưa ra quyết định tối ưu tại mỗi bước duyệt.

V. Thực Hành 2: Longest Increasing Subsequence (LIS) (35 phút)

1. Bài 2: Tìm độ dài dãy con tăng dần dài nhất (LIS) không liên tục.

- **Trạng thái DP: DP[i]** là **độ dài của dãy con tăng dần dài nhất kết thúc tại chỉ mục i** (phần tử **A[i]**).

- **Công thức Truy hồi:** Để tính **DP[i]**, ta phải tìm tất cả các chỉ mục **j < i** sao cho **A[j] < A[i]**. Sau đó, **DP[i]** sẽ là 1 cộng với giá trị **DP[j]** lớn nhất trong số các **j** đó.

DP[i] = 1 + max({DP[j] | 0 ≤ j < i và A[j] < A[i]} U {0})

- Độ phức tạp của thuật toán này là $O(N^2)$ (Do vòng lặp bên ngoài **i** và vòng lặp bên trong **j**).

1.1. Code Python:

```python
def longest_increasing_subsequence(arr):
    # Tìm độ dài của Dãy con Tăng dần Dài nhất (LIS) bằng Quy hoạch động (DP).
    # Độ phức tạp thời gian: O(N^2)
    # Độ phức tạp không gian: O(N)
    # Tham số:
    #   arr (list): Mảng số nguyên.
    # Trả về:
    #   int: Độ dài LIS.
    n = len(arr)
    if n == 0:
        return 0
    # Mảng DP: dp[i] lưu trữ độ dài của LIS kết thúc tại chỉ mục i.
    # Khởi tạo tất cả các giá trị là 1, vì mọi phần tử đều tự nó là một LIS dài 1.
    dp = [1] * n
    max_lis = 1 # Độ dài LIS toàn cục tìm thấy cho đến nay
    # Duyệt qua từng phần tử (i: chỉ mục kết thúc của LIS tiềm năng)
    for i in range(1, n):
        # Duyệt qua tất cả các phần tử trước đó (j < i)
        for j in range(i):
            # 1. Kiểm tra điều kiện tăng dần: arr[j] phải nhỏ hơn arr[i]
            if arr[j] < arr[i]:
                # 2. Áp dụng công thức truy hồi DP:
                # Nếu ta mở rộng LIS kết thúc tại j, độ dài mới là dp[j] + 1.
                # dp[i] sẽ là max của độ dài hiện tại hoặc độ dài mở rộng này.
                dp[i] = max(dp[i], dp[j] + 1)
        # 3. Cập nhật LIS lớn nhất toàn cục
        max_lis = max(max_lis, dp[i])
    return max_lis
# ---------------- CHẠY THỬ VÀ MINH HỌA O(N^2) ----------------
arr2 = [10, 9, 2, 5, 3, 7, 101, 18]
# LIS ví dụ: [2, 5, 7, 18] hoặc [2, 3, 7, 18]. Độ dài là 4.
print(f"Mang goc: {arr2}")
result_lis = longest_increasing_subsequence(arr2)
print(f"Do dai LIS (Python): {result_lis}")
# Ví dụ 2
arr3 = [0, 8, 4, 12, 2, 10, 6, 14, 1, 9]
# LIS ví dụ: [0, 2, 6, 9] hoặc [0, 4, 6, 9]. Độ dài là 4.
result_lis_2 = longest_increasing_subsequence(arr3)
print(f"\nMang goc: {arr3}")
print(f"Do dai LIS (Python): {result_lis_2}")
```

=> Kết quả sau khi chạy chương trình:

```
Mang goc: [10, 9, 2, 5, 3, 7, 101, 18]
Do dai LIS (Python): 4

Mang goc: [0, 8, 4, 12, 2, 10, 6, 14, 1, 9]
Do dai LIS (Python): 4
```

- Giải thích Code Python:

Tính toán: Để tính **DP[i]**, ta phải xem xét tất cả các **DP[j]** trước đó. Điều này tạo nên tính chồng chéo của bài toán con. Ví dụ, để tính LIS tại 7, ta phải biết LIS tối đa tại 2, 5, 3.

1.2. Code C++:

```cpp
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
using namespace std;         // Tiền tố std
```

```
int longest_increasing_subsequence(const vector<int>& arr)
{
    // Tim do dai cua Day con Tang dan Dai nhat (LIS) bang Quy hoach dong (DP).
    // Do phuc tap thoi gian: O(N^2)
    // Do phuc tap khong gian: O(N)
    int n = arr.size();
    if (n == 0)
    {
        return 0;
    }
    // Mang DP: dp[i] luu tru do dai cua LIS ket thuc tai chi muc i.
    // Khoi tao tat ca cac gia tri la 1, vi moi phan tu deu tu no la mot LIS dai 1.
    vector<int> dp(n, 1);
    int max_lis = 1; // Do dai LIS toan cuc tim thay cho den nay
    // Duyet qua tung phan tu (i: chi muc ket thuc cua LIS tiem nang)
    for (int i = 1; i < n; i++)
    {
        // Duyet qua tat ca cac phan tu truoc do (j < i)
        for (int j = 0; j < i; j++)
        {
            // 1. Kiem tra dieu kien tang dan: arr[j] phai nho hon arr[i]
            if (arr[j] < arr[i])
            {
                // 2. Ap dung cong thuc truy hoi DP:
                // Neu ta mo rong LIS ket thuc tai j, do dai moi la dp[j] + 1.
                // dp[i] se la max cua do dai hien tai hoac do dai mo rong nay.
                dp[i] = max(dp[i], dp[j] + 1);
            }
        }
        // 3. Cap nhat LIS lon nhat toan cuc
        max_lis = max(max_lis, dp[i]);
    }
    return max_lis;
}
// ----------------- CHAY THU VA MINH HOA O(N^2) -----------------
int main()
{
    vector<int> arr2 = {10, 9, 2, 5, 3, 7, 101, 18};
    // LIS vi du: [2, 5, 7, 18] hoac [2, 3, 7, 18]. Do dai la 4.
    cout << "Mang goc: ";
    for (int x : arr2)
    {
        cout << x << " ";
    }
    cout << endl;
    int result_lis = longest_increasing_subsequence(arr2);
    cout << "Do dai LIS (C++): " << result_lis << endl;
    // Vi du 2
    vector<int> arr3 = {0, 8, 4, 12, 2, 10, 6, 14, 1, 9};
    // LIS vi du: [0, 2, 6, 9] hoac [0, 4, 6, 9]. Do dai la 4.
    cout << "\nMang goc: ";
    for (int x : arr3)
    {
        cout << x << " ";
    }
    cout << endl;
    int result_lis_2 = longest_increasing_subsequence(arr3);
    cout << "Do dai LIS (C++): " << result_lis_2 << endl;
```

```
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
Mang goc: 10 9 2 5 3 7 101 18
Do dai LIS (C++): 4

Mang goc: 0 8 4 12 2 10 6 14 1 9
Do dai LIS (C++): 4
```

- Giải thích Code C++:

Tầm quan trọng của dp[i] = 1: Việc khởi tạo **dp[i]** bằng **1** là quan trọng vì ngay cả khi **A[i]** không thể mở rộng bất kỳ dãy con nào trước đó, nó vẫn tự tạo thành một dãy con tăng dần có độ dài **1**.

VI. Tổng kết & Bài tập Về nhà (10 phút)

1. Tóm tắt & So sánh DP Dãy con:

Bài toán	Tên thuật toán	Trạng thái DP (DP[i])	Công thức Truy hồi	Độ phức tạp
Max Subarray Sum	Kadane	Tổng **max** kết thúc tại **i**	max(A[i], DP[i-1] + A[i])	O(N)
LIS	DP Cơ bản	Độ dài **LIS** kết thúc tại **i**	1 + max(DP[j] | j < i, A[j] < A[i])	$O(N^2)$

Cả hai bài toán đều sử dụng lời giải tối ưu của các bài toán con trước đó (**i-1** hoặc **j < i**) để tìm lời giải tối ưu tại vị trí hiện tại **i**.

2. Bài tập Về nhà:

2.1. Bài tập 1 (LIS Invert): Tìm độ dài của **dãy con giảm dần dài nhất (LDS)** trong một mảng số nguyên. (Gợi ý: Chỉ cần thay đổi dấu bất đẳng thức trong công thức truy hồi của LIS).

2.1.1. Code Python:

```python
def longest_decreasing_subsequence(arr):
    # Tìm độ dài của Dãy con Giảm dần Dài nhất (LDS) bằng Quy hoạch động (DP).
    # Độ phức tạp thời gian: O(N^2)
    # Độ phức tạp không gian: O(N)
    # Tham số:
    # arr (list): Mảng số nguyên.
    # Trả về:
    # int: Độ dài LDS.
    n = len(arr)
    if n == 0:
        return 0
    # Mảng DP: dp[i] lưu trữ độ dài của LDS kết thúc tại chỉ mục i.
    dp = [1] * n
    max_lds = 1 # Độ dài LDS toàn cục tìm thấy cho đến nay
    # Vòng lặp ngoài (i: chỉ mục kết thúc)
    for i in range(1, n):
        # Vòng lặp trong (j: chỉ mục trước i)
        for j in range(i):
            # 1. Kiểm tra điều kiện giảm dần: arr[j] phải lớn hơn arr[i]
            if arr[j] > arr[i]:
                # 2. Áp dụng công thức truy hồi DP:
```

```
                # Mở rộng LDS kết thúc tại j: dp[j] + 1
                dp[i] = max(dp[i], dp[j] + 1)
        # 3. Cập nhật LDS lớn nhất toàn cục
        max_lds = max(max_lds, dp[i])
    return max_lds
# ---------------- CHẠY THỬ VÀ MINH HỌA O(N^2) ----------------
arr1 = [10, 9, 2, 5, 3, 7, 101, 18]
# LDS ví dụ: [10, 9, 2] (dài 3), [101, 18] (dài 2), [10, 9, 7, 3] (dài 4)
print(f"Mang goc: {arr1}")
result_lds_1 = longest_decreasing_subsequence(arr1)
print(f"Do dai LDS (Python): {result_lds_1} (LIS: 4)")
arr2 = [5, 4, 3, 2, 1]
# LDS: [5, 4, 3, 2, 1]. Độ dài là 5.
print(f"\nMang goc: {arr2}")
result_lds_2 = longest_decreasing_subsequence(arr2)
print(f"Do dai LDS (Python): {result_lds_2}")
arr3 = [1, 2, 3, 4, 5]
# LDS: [1] (hoặc bất kỳ phần tử đơn lẻ nào). Độ dài là 1.
print(f"\nMang goc: {arr3}")
result_lds_3 = longest_decreasing_subsequence(arr3)
print(f"Do dai LDS (Python): {result_lds_3}")
```

=> Kết quả sau khi chạy chương trình:

```
Mang goc: [10, 9, 2, 5, 3, 7, 101, 18]
Do dai LDS (Python): 4 (LIS: 4)

Mang goc: [5, 4, 3, 2, 1]
Do dai LDS (Python): 5

Mang goc: [1, 2, 3, 4, 5]
Do dai LDS (Python): 1
```

2.1.2. Code C++:

```
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
using namespace std;         // Tiền tố std
int longest_decreasing_subsequence(const vector<int>& arr)
{
    // Tim do dai cua Day con Giam dan Dai nhat (LDS) bang Quy hoach dong (DP).
    // Do phuc tap thoi gian: O(N^2)
    // Do phuc tap khong gian: O(N)
    int n = arr.size();
    if (n == 0)
    {
        return 0;
    }
    // Mang DP: dp[i] luu tru do dai cua LDS ket thuc tai chi muc i.
    vector<int> dp(n, 1);
    int max_lds = 1; // Do dai LDS toan cuc tim thay cho den nay
    // Vong lap ngoai (i: chi muc ket thuc)
    for (int i = 1; i < n; i++)
    {
        // Vong lap trong (j: chi muc truoc i)
        for (int j = 0; j < i; j++)
        {
            // 1. Kiem tra dieu kien giam dan: arr[j] phai lon hon arr[i]
            if (arr[j] > arr[i])
            {
                // 2. Ap dung cong thuc truy hoi DP:
```

```
                // Mo rong LDS ket thuc tai j: dp[j] + 1
                dp[i] = max(dp[i], dp[j] + 1);
            }
        }
        // 3. Cap nhat LDS lon nhat toan cuc
        max_lds = max(max_lds, dp[i]);
    }
    return max_lds;
}
// ----------------- CHAY THU VA MINH HOA O(N^2) -----------------
int main()
{
    vector<int> arr1 = {10, 9, 2, 5, 3, 7, 101, 18};
    // LDS vi du: [10, 9, 2] (dai 3), [101, 18] (dai 2), [10, 9, 7, 3] (dai 4)
    cout << "Mang goc: ";
    for (int x : arr1)
    {
        cout << x << " ";
    }
    cout << endl;
    int result_lds_1 = longest_decreasing_subsequence(arr1);
    cout << "Do dai LDS (C++): " << result_lds_1 << " (LIS: 4)" << endl;
    vector<int> arr2 = {5, 4, 3, 2, 1};
    // LDS: [5, 4, 3, 2, 1]. Do dai la 5.
    cout << "\nMang goc: ";
    for (int x : arr2)
    {
        cout << x << " ";
    }
    cout << endl;
    int result_lds_2 = longest_decreasing_subsequence(arr2);
    cout << "Do dai LDS (C++): " << result_lds_2 << endl;
    vector<int> arr3 = {1, 2, 3, 4, 5};
    // LDS: [1] (hoac bat ky phan tu don le nao). Do dai la 1.
    cout << "\nMang goc: ";
    for (int x : arr3)
    {
        cout << x << " ";
    }
    cout << endl;
    int result_lds_3 = longest_decreasing_subsequence(arr3);
    cout << "Do dai LDS (C++): " << result_lds_3 << endl;
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
Mang goc: 10 9 2 5 3 7 101 18
Do dai LDS (C++): 4 (LIS: 4)

Mang goc: 5 4 3 2 1
Do dai LDS (C++): 5

Mang goc: 1 2 3 4 5
Do dai LDS (C++): 1
```

2.2. Bài tập 2 (LIS & Kadane): Cho một mảng, tìm độ dài LIS, sau đó tính tổng lớn nhất của chính dãy con tăng dần đó. (Phức tạp hơn, cần lưu trữ thêm mảng tổng).

2.2.1. Code Python:

```python
def max_sum_longest_increasing_subsequence(arr):
    # Tìm độ dài LIS và Tổng lớn nhất của LIS đó.
    # Độ phức tạp thời gian: O(N^2)
    # Độ phức tạp không gian: O(N)
    # Trả về: (max_length, max_total_sum)
    n = len(arr)
    if n == 0:
        return (0, 0)
    # DP_len[i]: Độ dài LIS kết thúc tại i (Khởi tạo: 1)
    dp_len = [1] * n
    # DP_sum[i]: Tổng lớn nhất của LIS kết thúc tại i
    # Khởi tạo: Mọi dãy LIS dài 1 đều có tổng là chính giá trị arr[i]
    dp_sum = arr[:]
    # Biến theo dõi kết quả cuối cùng
    max_length = 1
    max_total_sum = arr[0]
    # Vòng lặp ngoài (i: chỉ mục kết thúc)
    for i in range(1, n):
        current_val = arr[i]
        # Vòng lặp trong (j: chỉ mục trước i)
        for j in range(i):
            if arr[j] < current_val:
                # Tính toán độ dài mới và tổng mới nếu mở rộng từ j
                new_len = dp_len[j] + 1
                new_sum = dp_sum[j] + current_val
                # --- LOGIC QUY HOẠCH ĐỘNG HAI CHIỀU ---
                # TRƯỜNG HỢP 1: TÌM THẤY DÃY DÀI HƠN
                if new_len > dp_len[i]:
                    dp_len[i] = new_len
                    dp_sum[i] = new_sum
                # TRƯỜNG HỢP 2: DÃY CÓ ĐỘ DÀI BẰNG NHAU
                elif new_len == dp_len[i]:
                    # Cập nhật tổng chỉ khi tổng mới lớn hơn tổng hiện tại
                    dp_sum[i] = max(dp_sum[i], new_sum)
        # Cập nhật kết quả toàn cục sau khi tính toán cho i
        if dp_len[i] > max_length:
            max_length = dp_len[i]
            max_total_sum = dp_sum[i]
        elif dp_len[i] == max_length:
            # Nếu độ dài bằng nhau, kiểm tra tổng lớn nhất
            max_total_sum = max(max_total_sum, dp_sum[i])
    return (max_length, max_total_sum)
# ----------------- CHẠY THỬ VÀ MINH HỌA O(N^2) -----------------
arr1 = [1, 101, 2, 3, 100, 4, 5]
# LIS dài 5: [1, 2, 3, 4, 5] (Tổng = 15)
# LIS dài 4: [1, 2, 3, 100] (Tổng = 106) -> KHÔNG ĐƯỢC CHỌN (vì không phải LIS dài
nhất)
# LIS dài 5: [1, 2, 3, 4, 5] hoặc [1, 2, 3, 100]
print(f"Mang goc: {arr1}")
length, total_sum = max_sum_longest_increasing_subsequence(arr1)
print(f"Do dai LIS: {length}") # 5
print(f"Tong lon nhat cua LIS: {total_sum}") # 111 (LIS: [1, 2, 3, 100, x] -> [1,
2, 3, 4, 5] -> [1, 2, 3, 100, 4, 5] --> [1, 2, 3, 100, 5] là 5 phần tử. 1+2+3+100+5
= 111)
arr2 = [3, 10, 4, 5, 11]
# LIS dài 4: [3, 4, 5, 11] (Tổng = 23)
# LIS dài 3: [3, 10, 11] (Tổng = 24) -> Bị loại vì không phải LIS dài nhất (4)
# LIS dài 4: [3, 4, 5, 11] (Tổng = 23)
```

```
print(f"\nMang goc: {arr2}")
length, total_sum = max_sum_longest_increasing_subsequence(arr2)
print(f"Do dai LIS: {length}") # 4
print(f"Tong lon nhat cua LIS: {total_sum}") # 23
```

=> Kết quả sau khi chạy chương trình:

```
Mang goc: [1, 101, 2, 3, 100, 4, 5]
Do dai LIS: 5
Tong lon nhat cua LIS: 15

Mang goc: [3, 10, 4, 5, 11]
Do dai LIS: 4
Tong lon nhat cua LIS: 23
```

2.2.2. Code C++:

```
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
using namespace std;         // Tiền tố std
pair<int, int> max_sum_longest_increasing_subsequence(const vector<int>& arr)
{
    // Tim do dai LIS va Tong lon nhat cua LIS do.
    // Do phuc tap thoi gian: O(N^2)
    // Do phuc tap khong gian: O(N)
    // Tra ve: (max_length, max_total_sum)
    int n = arr.size();
    if (n == 0)
    {
        return {0, 0};
    }
    // DP_len[i]: Do dai LIS ket thuc tai i (Khoi tao: 1)
    vector<int> dp_len(n, 1);
    // DP_sum[i]: Tong lon nhat cua LIS ket thuc tai i
    // Khoi tao: Moi day LIS dai 1 deu co tong la chinh gia tri arr[i]
    vector<int> dp_sum = arr;
    // Bien theo doi ket qua cuoi cung
    int max_length = 1;
    int max_total_sum = arr[0];
    // Vong lap ngoai (i: chi muc ket thuc)
    for (int i = 1; i < n; i++)
    {
        int current_val = arr[i];
        // Vong lap trong (j: chi muc truoc i)
        for (int j = 0; j < i; j++)
        {
            if (arr[j] < current_val)
            {
                // Tinh toan do dai moi va tong moi neu mo rong tu j
                int new_len = dp_len[j] + 1;
                int new_sum = dp_sum[j] + current_val;
                // --- LOGIC QUY HOACH DONG HAI CHIEU ---
                // TRUONG HOP 1: TIM THAY DAY DAI HON
                if (new_len > dp_len[i])
                {
                    dp_len[i] = new_len;
                    dp_sum[i] = new_sum;
                }
                // TRUONG HOP 2: DAY CO DO DAI BANG NHAU
                else if (new_len == dp_len[i])
                {
```

```
                    // Cap nhat tong chi khi tong moi lon hon tong hien tai
                    dp_sum[i] = max(dp_sum[i], new_sum);
                }
            }
        }
        // Cap nhat ket qua toan cuc sau khi tinh toan cho i
        if (dp_len[i] > max_length)
        {
            max_length = dp_len[i];
            max_total_sum = dp_sum[i];
        }
        else if (dp_len[i] == max_length)
        {
            // Neu do dai bang nhau, kiem tra tong lon nhat
            max_total_sum = max(max_total_sum, dp_sum[i]);
        }
    }
    return {max_length, max_total_sum};
}
// ----------------- CHAY THU VA MINH HOA O(N^2) -----------------
int main()
{
    vector<int> arr1 = {1, 101, 2, 3, 100, 4, 5};
    // LIS dai 5: [1, 2, 3, 4, 5] (Tong = 15)
    // LIS dai 4: [1, 2, 3, 100] (Tong = 106) -> KHONG DUOC CHON (vi khong phai LIS
dai nhat)
    cout << "Mang goc: ";
    for (int x : arr1)
    {
        cout << x << " ";
    }
    cout << endl;
    auto result1 = max_sum_longest_increasing_subsequence(arr1);
    cout << "Do dai LIS: " << result1.first << endl;
    cout << "Tong lon nhat cua LIS: " << result1.second << endl;
    vector<int> arr2 = {3, 10, 4, 5, 11};
    // LIS dai 4: [3, 4, 5, 11] (Tong = 23)
    // LIS dai 3: [3, 10, 11] (Tong = 24) -> Bi loai vi khong phai LIS dai nhat
    cout << "\nMang goc: ";
    for (int x : arr2)
    {
        cout << x << " ";
    }
    cout << endl;
    auto result2 = max_sum_longest_increasing_subsequence(arr2);
    cout << "Do dai LIS: " << result2.first << endl;
    cout << "Tong lon nhat cua LIS: " << result2.second << endl;
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
Mang goc: 1 101 2 3 100 4 5
Do dai LIS: 5
Tong lon nhat cua LIS: 15

Mang goc: 3 10 4 5 11
Do dai LIS: 4
Tong lon nhat cua LIS: 23
```

2.3. Bài tập 3 (Thao tác LIS O(N log N)): Nghiên cứu và cài đặt thuật toán LIS tối ưu hơn bằng cách sử dụng **Tìm kiếm nhị phân (Binary Search)** để giảm độ phức tạp xuống O(N log N).

2.3.1. Code Python:

```
from bisect import bisect_left  # Giúp tìm kiếm nhị phân hiệu quả
def longest_increasing_subsequence_nlogn(arr):
    # Tìm độ dài của Dãy con Tăng dần Dài nhất (LIS) bằng thuật toán DP tối ưu.
    # Sử dụng Tìm kiếm nhị phân (Binary Search).
    # Độ phức tạp thời gian: O(N log N)
    # Độ phức tạp không gian: O(N)
    # Tham số:
    # arr (list): Mảng số nguyên.
    # Trả về:
    # int: Độ dài LIS (bằng len(tails)).
    if not arr:
        return 0
    # Mảng tails: Lưu trữ giá trị kết thúc nhỏ nhất của các LIS có độ dài tương
ứng.
    # tails[i] là giá trị kết thúc nhỏ nhất của LIS có độ dài i + 1.
    tails = []
    for num in arr:
        # Sử dụng bisect_left để tìm chỉ mục (idx) của phần tử đầu tiên
        # trong tails mà lớn hơn hoặc bằng num.
        # Thao tác này là O(log N).
        idx = bisect_left(tails, num)
        # TRƯỜNG HỢP 1: num lớn hơn tất cả các phần tử trong tails (idx ==
len(tails))
        # -> Mở rộng LIS dài nhất hiện tại.
        if idx == len(tails):
            tails.append(num)
        # TRƯỜNG HỢP 2: num nhỏ hơn hoặc bằng một phần tử trong tails
        # -> Thay thế tails[idx] bằng num.
        # Điều này tạo ra một LIS có độ dài (idx + 1) với giá trị kết thúc nhỏ hơn,
        # làm tăng cơ hội mở rộng dãy con này trong tương lai.
        else:
            tails[idx] = num
    # Độ dài LIS chính là độ dài của mảng tails
    return len(tails)
# ----------------- CHẠY THỬ VÀ MINH HỌA O(N log N) -----------------
arr1 = [10, 9, 2, 5, 3, 7, 101, 18]
# Kết quả mong đợi: 4 (LIS: [2, 5, 7, 18] hoặc [2, 3, 7, 18])
print(f"Mang goc: {arr1}")
result_nlogn_1 = longest_increasing_subsequence_nlogn(arr1)
print(f"Do dai LIS (O(N log N)): {result_nlogn_1}")
arr2 = [0, 8, 4, 12, 2, 10, 6, 14]
# Kết quả mong đợi: 5 (LIS: [0, 4, 6, 14] hoặc [0, 2, 6, 14]) -> Lỗi LIS là 4,
không phải 5
# LIS: [0, 4, 10, 14] (dài 4) hoặc [0, 2, 6, 14] (dài 4)
print(f"\nMang goc: {arr2}")
result_nlogn_2 = longest_increasing_subsequence_nlogn(arr2)
print(f"Do dai LIS (O(N log N)): {result_nlogn_2}") # 4
```

=> Kết quả sau khi chạy chương trình:

```
Mang goc: [10, 9, 2, 5, 3, 7, 101, 18]
Do dai LIS (O(N log N)): 4

Mang goc: [0, 8, 4, 12, 2, 10, 6, 14]
Do dai LIS (O(N log N)): 4
```

2.3.2. Code C++:

```
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
#include <algorithm>         // lower_bound
using namespace std;         // Tiền tố std
int longest_increasing_subsequence_nlogn(const vector<int>& arr)
{
    // Tim do dai cua Day con Tang dan Dai nhat (LIS) bang thuat toan DP toi uu.
    // Su dung Tim kiem nhi phan (Binary Search).
    // Do phuc tap thoi gian: O(N log N)
    // Do phuc tap khong gian: O(N)
    // Tham so:
    // arr (vector): Mang so nguyen.
    // Tra ve:
    // int: Do dai LIS (bang do dai cua mang tails).
    if (arr.empty())
    {
        return 0;
    }
    // Mang tails: Luu tru gia tri ket thuc nho nhat cua cac LIS co do dai tuong
ung.
    // tails[i] la gia tri ket thuc nho nhat cua LIS co do dai i + 1.
    vector<int> tails;
    for (int num : arr)
    {
        // Su dung lower_bound de tim chi muc (idx) cua phan tu dau tien
        // trong tails ma lon hon hoac bang num.
        // Thao tac nay la O(log N).
        int idx = lower_bound(tails.begin(), tails.end(), num) - tails.begin();
        // TRUONG HOP 1: num lon hon tat ca cac phan tu trong tails (idx ==
tails.size())
        // -> Mo rong LIS dai nhat hien tai.
        if (idx == tails.size())
        {
            tails.push_back(num);
        }
        // TRUONG HOP 2: num nho hon hoac bang mot phan tu trong tails
        // -> Thay the tails[idx] bang num.
        // Dieu nay tao ra mot LIS co do dai (idx + 1) voi gia tri ket thuc nho
hon,
        // lam tang co hoi mo rong day con nay trong tuong lai.
        else
        {
            tails[idx] = num;
        }
    }
    // Do dai LIS chinh la do dai cua mang tails
    return tails.size();
}
// ----------------- CHAY THU VA MINH HOA O(N log N) -----------------
int main()
{
```

```
    vector<int> arr1 = {10, 9, 2, 5, 3, 7, 101, 18};
    // Ket qua mong doi: 4 (LIS: [2, 5, 7, 18] hoac [2, 3, 7, 18])
    cout << "Mang goc: ";
    for (int x : arr1)
    {
        cout << x << " ";
    }
    cout << endl;
    int result1 = longest_increasing_subsequence_nlogn(arr1);
    cout << "Do dai LIS (O(N log N)): " << result1 << endl;
    vector<int> arr2 = {0, 8, 4, 12, 2, 10, 6, 14};
    // LIS: [0, 4, 10, 14] (dai 4) hoac [0, 2, 6, 14] (dai 4)
    cout << "\nMang goc: ";
    for (int x : arr2)
    {
        cout << x << " ";
    }
    cout << endl;
    int result2 = longest_increasing_subsequence_nlogn(arr2);
    cout << "Do dai LIS (O(N log N)): " << result2 << endl;
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
Mang goc: 10 9 2 5 3 7 101 18
Do dai LIS (O(N log N)): 4

Mang goc: 0 8 4 12 2 10 6 14
Do dai LIS (O(N log N)): 4
```

BÀI 26. TÌM DÃY CON TĂNG DÀI NHẤT (LIS)

I. Mục Tiêu:

- **Kiến thức:** HS nắm vững bài toán LIS, hiểu rõ công thức truy hồi $O(N^2)$ và nguyên lý tối ưu hóa bằng Tìm kiếm nhị phân O(N log N).
- **Kỹ năng:** HS cài đặt được cả hai phiên bản thuật toán LIS trong Python và C++.
- **Tư duy:** HS phân tích được trade-off giữa độ phức tạp thời gian và không gian trong các phương pháp giải DP.

II. Phân Bổ Thời Gian (90 Phút):

Hoạt Động	Nội Dung	Thời gian (Phút)
Ôn tập & Giới thiệu	Định nghĩa LIS, DP $O(N^2)$ (tóm tắt)	10
Thực hành 1	Bài 1. LIS cơ bản $O(N^2)$ (Code mẫu chi tiết)	30
Lý thuyết 2	Nguyên lý Tối ưu hóa O(N log N) (Mảng tails)	15
Thực hành 2	Bài 2. LIS tối ưu O(N log N) (Code mẫu chi tiết)	30
Tổng kết	So sánh hai phương pháp, Bài tập về nhà	5
Tổng cộng		**90**

III. Lý Thuyết 1: LIS Cơ bản (O(N^2)) (10 phút)

1. Định nghĩa LIS:

Dãy con Tăng dần Dài nhất (LIS) là dãy con được tạo thành bằng cách chọn các phần tử trong mảng gốc sao cho chúng xuất hiện theo đúng thứ tự ban đầu và giá trị của chúng tăng dần. Các phần tử không nhất thiết phải liền kề nhau.

2. Trạng thái và Công thức Truy hồi DP:

- **Trạng thái DP:** DP[i] là **độ dài của dãy con tăng dần dài nhất kết thúc tại chỉ mục i** (bao gồm cả A[i]).
- **Công thức Truy hồi:**

$$DP[i] = 1 + \max(\{DP[j] \mid 0 \le j < i \text{ và } A[j] < A[i]\} \cup \{0\})$$

IV. Thực Hành 1: LIS O(N^2) (30 phút)

1. Bài 1: Cài đặt thuật toán LIS DP O(N^2).

1.1. Code Python:

```
def longest_increasing_subsequence_n2(arr):
    # Tìm độ dài của Dãy con Tăng dần Dài nhất (LIS) bằng Quy hoạch động (DP)
O(N^2).
    # Tham số:
    # arr (list): Mảng số nguyên.
    # Trả về:
    # int: Độ dài LIS.
    n = len(arr)
    if n == 0:
        return 0
    # Mảng DP: dp[i] lưu trữ độ dài của LIS kết thúc tại chỉ mục i.
    # Khởi tạo tất cả các giá trị là 1 (LIS ngắn nhất là chính nó).
    dp = [1] * n
    max_lis = 1 # Độ dài LIS lớn nhất toàn cục
    # Vòng lặp ngoài (i: chỉ mục kết thúc hiện tại, từ 1 đến n-1)
    for i in range(1, n):
        # Vòng lặp trong (j: chỉ mục trước i, từ 0 đến i-1)
        # Mục đích: Tìm LIS tối ưu (dp[j]) để mở rộng
        for j in range(i):
            # 1. Điều kiện kiểm tra: arr[j] phải nhỏ hơn arr[i]
            # (đảm bảo tính chất tăng dần)
            if arr[j] < arr[i]:
                # 2. Công thức Truy hồi DP:
                # Cập nhật dp[i] bằng giá trị lớn nhất giữa giá trị hiện tại
                # và (độ dài LIS kết thúc tại j) + 1
                dp[i] = max(dp[i], dp[j] + 1)
        # 3. Cập nhật LIS lớn nhất toàn cục
        max_lis = max(max_lis, dp[i])
    return max_lis
# ----------------- CHẠY THỬ VÀ MINH HỌA -----------------
arr1 = [10, 9, 2, 5, 3, 7, 101, 18]
# LIS ví dụ: [2, 5, 7, 18] hoặc [2, 3, 7, 18]. Độ dài là 4.
print(f"Mang goc: {arr1}")
result_n2 = longest_increasing_subsequence_n2(arr1)
print(f"Do dai LIS (O(N^2)): {result_n2}")
arr2 = [3, 10, 2, 1, 20]
# LIS: [3, 10, 20] hoặc [3, 4, 5, 6, 7]
# LIS: [3, 10, 20]
print(f"\nMang goc: {arr2}")
result_n2_2 = longest_increasing_subsequence_n2(arr2)
```

```
print(f"Do dai LIS (O(N^2)): {result_n2_2}") # 3
```

=> Kết quả sau khi chạy chương trình:

```
Mang goc: [10, 9, 2, 5, 3, 7, 101, 18]
Do dai LIS (O(N^2)): 4

Mang goc: [3, 10, 2, 1, 20]
Do dai LIS (O(N^2)): 3
```

- Giải thích Code Python:

Độ phức tạp: Vòng lặp lồng nhau (i từ 1 đến N, j từ 0 đến i) dẫn đến $O(N^2)$. DP[i] được tính bằng cách kiểm tra tất cả các **j** trước đó để tìm ra LIS tối ưu nhất có thể mở rộng tới **i**.

1.2. Code C++:

```
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
using namespace std;         // Tiền tố std
int longest_increasing_subsequence_n2(const vector<int>& arr)
{
    // Tim do dai cua Day con Tang dan Dai nhat (LIS) bang Quy hoach dong (DP)
O(N^2).
    // Tham so:
    // arr (vector): Mang so nguyen.
    // Tra ve:
    // int: Do dai LIS.
    int n = arr.size();
    if (n == 0)
    {
        return 0;
    }
    // Mang DP: dp[i] luu tru do dai cua LIS ket thuc tai chi muc i.
    // Khoi tao tat ca cac gia tri la 1 (LIS ngan nhat la chinh no).
    vector<int> dp(n, 1);
    int max_lis = 1; // Do dai LIS lon nhat toan cuc
    // Vong lap ngoai (i: chi muc ket thuc hien tai, tu 1 den n-1)
    for (int i = 1; i < n; i++)
    {
        // Vong lap trong (j: chi muc truoc i, tu 0 den i-1)
        // Muc dich: Tim LIS toi uu (dp[j]) de mo rong
        for (int j = 0; j < i; j++)
        {
            // 1. Dieu kien kiem tra: arr[j] phai nho hon arr[i]
            // (dam bao tinh chat tang dan)
            if (arr[j] < arr[i])
            {
                // 2. Cong thuc Truy hoi DP:
                // Cap nhat dp[i] bang gia tri lon nhat giua gia tri hien tai
                // va (do dai LIS ket thuc tai j) + 1
                dp[i] = max(dp[i], dp[j] + 1);
            }
        }
        // 3. Cap nhat LIS lon nhat toan cuc
        max_lis = max(max_lis, dp[i]);
    }
    return max_lis;
}
// ---------------- CHAY THU VA MINH HOA ----------------
int main()
```

```
{
    vector<int> arr1 = {10, 9, 2, 5, 3, 7, 101, 18};
    // LIS vi du: [2, 5, 7, 18] hoac [2, 3, 7, 18]. Do dai la 4.
    cout << "Mang goc: ";
    for (int x : arr1)
    {
        cout << x << " ";
    }
    cout << endl;
    int result_n2 = longest_increasing_subsequence_n2(arr1);
    cout << "Do dai LIS (O(N^2)): " << result_n2 << endl;
    vector<int> arr2 = {3, 10, 2, 1, 20};
    // LIS: [3, 10, 20]
    cout << "\nMang goc: ";
    for (int x : arr2)
    {
        cout << x << " ";
    }
    cout << endl;
    int result_n2_2 = longest_increasing_subsequence_n2(arr2);
    cout << "Do dai LIS (O(N^2)): " << result_n2_2 << endl; // 3
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
Mang goc: 10 9 2 5 3 7 101 18
Do dai LIS (O(N^2)): 4

Mang goc: 3 10 2 1 20
Do dai LIS (O(N^2)): 3
```

- Giải thích Code C++:

Sử dụng std::vector: Tương tự Python, mảng **dp** được khởi tạo với giá trị 1. Logic của hai vòng lặp lồng nhau thực hiện chính xác công thức truy hồi **DP**.

V. Lý Thuyết 2: LIS Tối ưu (O(N log N)) (15 phút)

1. Nguyên lý Tối ưu hóa:

Thay vì duyệt tất cả các **j < i** để tìm **max(DP[j])**, ta có thể sử dụng mảng phụ **tails** và tìm kiếm nhị phân để giảm thời gian tìm kiếm từ O(N) xuống O(log N).

2. Mảng tails (Giá trị kết thúc nhỏ nhất):

- Mảng **tails** lưu trữ **giá trị kết thúc nhỏ nhất** của tất cả các dãy con tăng dần có độ dài tương ứng.
- **Ví dụ:** Nếu tails = [2, 5, 8], điều này có nghĩa:
  + Có một LIS dài 1 kết thúc bằng 2 (nhỏ nhất).
  + Có một LIS dài 2 kết thúc bằng 5 (nhỏ nhất).
  + Có một LIS dài 3 kết thúc bằng 8 (nhỏ nhất).
- Mảng **tails** luôn **tăng dần nghiêm ngặt**.

3. Quy tắc Tối ưu hóa:

Khi duyệt qua A[i]:

+ Nếu A[i] lớn hơn tails cuối cùng: Mở rộng LIS dài nhất. => Thêm A[i] vào cuối **tails**. (O(1))

+ **Ngược lại:** Tìm kiếm nhị phân phần tử **nhỏ nhất lớn hơn hoặc bằng A[i]** trong tails và thay thế nó bằng A[i]. (Thay thế này làm cho dãy con tăng dần có độ dài tương ứng tốt hơn). (O(log N))

VI. Thực Hành 2: LIS $O(N log N) (30 phút)

1. Bài 2: Cài đặt thuật toán LIS tối ưu O(N log N) bằng Binary Search.

1.1. Code Python:

```
from bisect import bisect_left  # Giúp tìm kiếm nhị phân tốt hơn
def longest_increasing_subsequence_nlogn(arr):
    # Tìm độ dài của Dãy con Tăng dần Dài nhất (LIS) bằng thuật toán tối ưu.
    # Sử dụng Tìm kiếm nhị phân (bisect_left).
    # Độ phức tạp thời gian: O(N log N)
    # Độ phức tạp không gian: O(N)
    # Tham số:
    # arr (list): Mảng số nguyên.
    # Trả về:
    # int: Độ dài LIS (bằng len(tails)).
    if not arr:
        return 0
    # Mảng tails: Lưu trữ giá trị kết thúc nhỏ nhất của các LIS có độ dài tương
ứng.
    # Mảng này luôn được duy trì ở trạng thái tăng dần.
    tails = []
    for num in arr:
        # Sử dụng bisect_left: Tìm chỉ mục (idx) của phần tử đầu tiên
        # trong tails mà lớn hơn hoặc bằng num. (Thao tác O(log N)).
        idx = bisect_left(tails, num)
        # TRƯỜNG HỢP 1: Mở rộng LIS (idx == len(tails))
        # num lớn hơn tất cả các giá trị kết thúc hiện có.
        # Nó mở rộng LIS dài nhất hiện tại thêm 1 đơn vị.
        if idx == len(tails):
            tails.append(num)
        # TRƯỜNG HỢP 2: Cập nhật giá trị kết thúc
        # tails[idx] là phần tử nhỏ nhất lớn hơn hoặc bằng num.
        # Thay thế tails[idx] bằng num. Điều này không làm thay đổi độ dài LIS,
        # nhưng nó tạo ra một LIS cùng độ dài nhưng kết thúc bằng giá trị nhỏ hơn,
        # làm tăng cơ hội mở rộng dãy con này trong tương lai.
        else:
            tails[idx] = num
    # Độ dài LIS chính là độ dài của mảng tails
    return len(tails)
# ----------------- CHẠY THỬ VÀ MINH HỌA O(N log N) -----------------
arr2 = [10, 9, 2, 5, 3, 7, 101, 18]
# LIS ví dụ: [2, 5, 7, 18]. Độ dài: 4.
print(f"Mang goc: {arr2}")
result_nlogn = longest_increasing_subsequence_nlogn(arr2)
print(f"Do dai LIS (O(N log N)): {result_nlogn}")
arr3 = [4, 1, 5, 2, 6, 3]
# Quá trình tails:
# 4 -> [4]
# 1 -> [1] (Thay thế 4)
# 5 -> [1, 5]
# 2 -> [1, 2] (Thay thế 5)
# 6 -> [1, 2, 6]
# 3 -> [1, 2, 3] (Thay thế 6)
# Cuối cùng: [1, 2, 3]. Độ dài: 3. (LIS: [1, 2, 3] hoặc [1, 5, 6])
print(f"\nMang goc: {arr3}")
```

```
result_nlogn_2 = longest_increasing_subsequence_nlogn(arr3)
print(f"Do dai LIS (O(N log N)): {result_nlogn_2}") # 3
```

=> Kết quả sau khi chạy chương trình:

```
Mang goc: [10, 9, 2, 5, 3, 7, 101, 18]
Do dai LIS (O(N log N)): 4

Mang goc: [4, 1, 5, 2, 6, 3]
Do dai LIS (O(N log N)): 3
```

- Giải thích Code Python:

Hàm bisect_left: Python cung cấp sẵn hàm này. Nó trả về vị trí chèn để duy trì thứ tự tăng dần. Nếu vị trí chèn là cuối mảng, ta **append**. Nếu không, ta **tails[idx] = num**. Tổng cộng **N lần lặp x O(log N) = O(N log N)**.

1.2. Code C++:

```
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
#include <algorithm>         // lower_bound tương đương bisect_left trong Python
using namespace std;         // Tiền tố std
int longest_increasing_subsequence_nlogn(const vector<int>& arr)
{
    // Tim do dai cua Day con Tang dan Dai nhat (LIS) bang thuat toan toi uu.
    // Su dung Tim kiem nhi phan (lower_bound).
    // Do phuc tap thoi gian: O(N log N)
    // Do phuc tap khong gian: O(N)
    // Tham so:
    // arr (vector): Mang so nguyen.
    // Tra ve:
    // int: Do dai LIS (bang kich thuoc cua mang tails).
    if (arr.empty())
    {
        return 0;
    }
    // Mang tails: Luu tru gia tri ket thuc nho nhat cua cac LIS co do dai tuong
ung.
    // Mang nay luon duoc duy tri o trang thai tang dan.
    vector<int> tails;
    for (int num : arr)
    {
        // Su dung lower_bound: Tim chi muc (idx) cua phan tu dau tien
        // trong tails ma lon hon hoac bang num. (Thao tac O(log N)).
        auto it = lower_bound(tails.begin(), tails.end(), num);
        int idx = it - tails.begin();
        // TRUONG HOP 1: Mo rong LIS (idx == tails.size())
        // num lon hon tat ca cac gia tri ket thuc hien co.
        // No mo rong LIS dai nhat hien tai them 1 don vi.
        if (idx == tails.size())
        {
            tails.push_back(num);
        }
        // TRUONG HOP 2: Cap nhat gia tri ket thuc
        // Thay the tails[idx] bang num.
        // Khong thay doi do dai LIS, nhung lam gia tri ket thuc nho hon,
        // tang kha nang mo rong trong tuong lai.
        else
        {
            tails[idx] = num;
```

```
        }
    }
    // Do dai LIS chinh la do dai cua mang tails
    return tails.size();
}
// ---------------- CHAY THU VA MINH HOA O(N log N) ----------------
int main()
{
    vector<int> arr2 = {10, 9, 2, 5, 3, 7, 101, 18};
    // LIS vi du: [2, 5, 7, 18]. Do dai: 4.
    cout << "Mang goc: ";
    for (int x : arr2)
    {
        cout << x << " ";
    }
    cout << endl;
    int result_nlogn = longest_increasing_subsequence_nlogn(arr2);
    cout << "Do dai LIS (O(N log N)): " << result_nlogn << endl;
    vector<int> arr3 = {4, 1, 5, 2, 6, 3};
    // Qua trinh tails:
    // 4 -> [4]
    // 1 -> [1]
    // 5 -> [1, 5]
    // 2 -> [1, 2]
    // 6 -> [1, 2, 6]
    // 3 -> [1, 2, 3]
    // Ket qua cuoi cung: Do dai = 3
    cout << "\nMang goc: ";
    for (int x : arr3)
    {
        cout << x << " ";
    }
    cout << endl;
    int result_nlogn_2 = longest_increasing_subsequence_nlogn(arr3);
    cout << "Do dai LIS (O(N log N)): " << result_nlogn_2 << endl;
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
Mang goc: 10 9 2 5 3 7 101 18
Do dai LIS (O(N log N)): 4

Mang goc: 4 1 5 2 6 3
Do dai LIS (O(N log N)): 3
```

- Giải thích Code C++:

Hàm lower_bound: C++ sử dụng **lower_bound** (tương đương **bisect_left** của Python) để tìm vị trí. **it** là một con trỏ (**iterator**); ***it = num** thay thế giá trị tại vị trí đó.

VII. Tổng kết & Bài tập Về nhà (5 phút)

1. Tóm tắt:

Phương pháp	Độ phức tạp Thời gian	Độ phức tạp Không gian	Ghi chú
DP Cơ bản	$O(N^2)$	O(N)	Dễ cài đặt, dễ tìm lại dãy con LIS.
DP Tối ưu	O(N log N)	O(N)	Nhanh hơn cho N lớn, khó tìm lại dãy con LIS.

2. Bài tập Về nhà:

2.1. Bài tập 1 (Xây dựng LIS): Sử dụng phương pháp $O(N^2)$, xây dựng lại không chỉ độ dài mà còn in ra một trong các dãy con tăng dần dài nhất. (Gợi ý: Cần thêm một mảng **parent** hoặc **predecessor** để lưu chỉ mục của phần tử trước đó).

2.1.1. Code Python:

```python
def construct_longest_increasing_subsequence(arr):
    # Sử dụng DP O(N^2) để tìm độ dài LIS và xây dựng lại một trong các dãy con đó.
    # Tham số:
    # arr (list): Mảng số nguyên.
    # Trả về:
    # tuple: (list LIS, int max_length)
    n = len(arr)
    if n == 0:
        return ([], 0)
    # 1. Khởi tạo các mảng DP
    # dp_len[i]: Độ dài LIS kết thúc tại i.
    dp_len = [1] * n
    # prev_index[i]: Chỉ mục của phần tử đứng ngay trước arr[i] trong LIS đó.
    # Giá trị -1 đại diện cho phần tử đầu tiên của dãy con (không có tiền nhiệm).
    prev_index = [-1] * n
    # Biến theo dõi kết quả toàn cục
    max_lis = 1
    end_index = 0 # Chỉ mục cuối cùng của LIS dài nhất hiện tại
    # 2. Quá trình tính toán DP (O(N^2))
    for i in range(n):
        for j in range(i):
            # Điều kiện LIS và tìm thấy dãy dài hơn
            if arr[j] < arr[i]:
                # Nếu mở rộng LIS của j tạo ra dãy dài hơn
                if dp_len[j] + 1 > dp_len[i]:
                    # Cập nhật độ dài LIS kết thúc tại i
                    dp_len[i] = dp_len[j] + 1
                    # Ghi lại chỉ mục j là tiền nhiệm của i
                    prev_index[i] = j
        # Cập nhật kết quả toàn cục
        if dp_len[i] > max_lis:
            max_lis = dp_len[i]
            end_index = i # Lưu chỉ mục cuối của LIS dài nhất mới
    # 3. Quá trình Truy vết (Reconstruction)
    lis_sequence = []
    # Bắt đầu từ chỉ mục cuối của LIS dài nhất
    current_idx = end_index
    # Lặp ngược lại theo mảng prev_index cho đến khi gặp -1
    while current_idx != -1:
        lis_sequence.append(arr[current_idx])
        current_idx = prev_index[current_idx]
    # Dãy con được xây dựng ngược, nên cần đảo ngược lại
    lis_sequence.reverse()
    return (lis_sequence, max_lis)
# ----------------- CHẠY THỬ VÀ MINH HỌA -----------------
arr1 = [10, 9, 2, 5, 3, 7, 101, 18]
# LIS: [2, 5, 7, 18] (hoặc [2, 3, 7, 18])
lis_seq_1, length_1 = construct_longest_increasing_subsequence(arr1)
print(f"Mang goc: {arr1}")
print(f"Do dai LIS: {length_1}")
print(f"Day con LIS: {lis_seq_1}")
```

```
# Dãy con LIS: [2, 5, 7, 18]
arr2 = [3, 10, 2, 1, 20]
# LIS: [3, 10, 20]
lis_seq_2, length_2 = construct_longest_increasing_subsequence(arr2)
print(f"\nMang goc: {arr2}")
print(f"Do dai LIS: {length_2}")
print(f"Day con LIS: {lis_seq_2}")
```

=> Kết quả sau khi chạy chương trình:

```
Mang goc: [10, 9, 2, 5, 3, 7, 101, 18]
Do dai LIS: 4
Day con LIS: [2, 5, 7, 101]

Mang goc: [3, 10, 2, 1, 20]
Do dai LIS: 3
Day con LIS: [3, 10, 20]
```

2.1.2. Code C++:

```
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
using namespace std;         // Tiền tố std
pair<vector<int>, int> construct_longest_increasing_subsequence(const vector<int>&
arr)
{
    // Su dung DP O(N^2) de tim do dai LIS va xay dung lai mot trong cac day con
do.
    // Tham so:
    // arr (vector): Mang so nguyen.
    // Tra ve:
    // pair: (vector LIS, int max_length)
    int n = arr.size();
    if (n == 0)
    {
        return { {}, 0 };
    }
    // 1. Khoi tao cac mang DP
    // dp_len[i]: Do dai LIS ket thuc tai i.
    vector<int> dp_len(n, 1);
    // prev_index[i]: Chi muc cua phan tu dung ngay truoc arr[i] trong LIS do.
    // Gia tri -1 dai dien cho phan tu dau tien cua day con (khong co tien nhiem).
    vector<int> prev_index(n, -1);
    // Bien theo doi ket qua toan cuc
    int max_lis = 1;
    int end_index = 0; // Chi muc cuoi cung cua LIS dai nhat hien tai
    // 2. Qua trinh tinh toan DP (O(N^2))
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < i; j++)
        {
            // Dieu kien LIS va tim thay day dai hon
            if (arr[j] < arr[i])
            {
                // Neu mo rong LIS cua j tao ra day dai hon
                if (dp_len[j] + 1 > dp_len[i])
                {
                    // Cap nhat do dai LIS ket thuc tai i
                    dp_len[i] = dp_len[j] + 1;
                    // Ghi lai chi muc j la tien nhiem cua i
```

```
                    prev_index[i] = j;
                }
            }
        }
        // Cap nhat ket qua toan cuc
        if (dp_len[i] > max_lis)
        {
            max_lis = dp_len[i];
            end_index = i; // Luu chi muc cuoi cua LIS dai nhat moi
        }
    }
    // 3. Qua trinh Truy vet (Reconstruction)
    vector<int> lis_sequence;
    // Bat dau tu chi muc cuoi cua LIS dai nhat
    int current_idx = end_index;
    // Lap nguoc lai theo mang prev_index cho den khi gap -1
    while (current_idx != -1)
    {
        lis_sequence.push_back(arr[current_idx]);
        current_idx = prev_index[current_idx];
    }
    // Day con duoc xay dung nguoc, nen can dao nguoc lai
    reverse(lis_sequence.begin(), lis_sequence.end());
    return
    {
        lis_sequence, max_lis
    };
}
// ----------------- CHAY THU VA MINH HOA -----------------
int main()
{
    vector<int> arr1 = {10, 9, 2, 5, 3, 7, 101, 18};
    // LIS: [2, 5, 7, 18] (hoac [2, 3, 7, 18])
    auto result1 = construct_longest_increasing_subsequence(arr1);
    vector<int> lis_seq_1 = result1.first;
    int length_1 = result1.second;
    cout << "Mang goc: ";
    for (int x : arr1) cout << x << " ";
    cout << endl;
    cout << "Do dai LIS: " << length_1 << endl;
    cout << "Day con LIS: ";
    for (int x : lis_seq_1) cout << x << " ";
    cout << endl;
    vector<int> arr2 = {3, 10, 2, 1, 20};
    // LIS: [3, 10, 20]
    auto result2 = construct_longest_increasing_subsequence(arr2);
    vector<int> lis_seq_2 = result2.first;
    int length_2 = result2.second;
    cout << "\nMang goc: ";
    for (int x : arr2) cout << x << " ";
    cout << endl;
    cout << "Do dai LIS: " << length_2 << endl;
    cout << "Day con LIS: ";
    for (int x : lis_seq_2) cout << x << " ";
    cout << endl;
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
Mang goc: 10 9 2 5 3 7 101 18
Do dai LIS: 4
Day con LIS: 2 5 7 101

Mang goc: 3 10 2 1 20
Do dai LIS: 3
Day con LIS: 3 10 20
```

2.2. Bài tập 2 (LIS 2D): Cho một danh sách các cặp (chiều rộng, chiều cao), tìm chuỗi lớn nhất các cặp mà mỗi cặp sau đều lớn hơn cả chiều rộng và chiều cao của cặp trước đó. (Ví dụ: (w1, h1) theo sau (w2, h2) nếu w1 > w2 và h1 > h2).

2.2.1. Code Python:

```python
def longest_chain_of_pairs(pairs):
    # Tìm độ dài chuỗi lớn nhất các cặp (w, h) sao cho mỗi cặp sau đều lớn
    # hơn cả chiều rộng và chiều cao của cặp trước đó.
    # Độ phức tạp: O(N log N) cho sắp xếp, O(N^2) cho DP -> Tổng O(N^2).
    # Tham số:
    # pairs (list of tuples): Danh sách các cặp (chiều rộng, chiều cao).
    # Trả về:
    # int: Độ dài chuỗi lớn nhất.
    n = len(pairs)
    if n == 0:
        return 0
    # BƯỚC 1: Sắp xếp danh sách các cặp (Preprocessing)
    # Sắp xếp theo chiều rộng (w) tăng dần. Nếu w bằng nhau, sắp xếp theo chiều cao
(h) tăng dần.
    # Việc này đảm bảo khi ta xét cặp i, các cặp j trước đó đã được xử lý
    # và ta chỉ cần quan tâm đến điều kiện h_j < h_i.
    pairs.sort()
    print(f"Danh sach cac cap da sap xep: {pairs}")
    # BƯỚC 2: Áp dụng Quy hoạch động (DP O(N^2))
    # dp[i] là độ dài chuỗi lớn nhất kết thúc tại pairs[i].
    dp = [1] * n
    max_chain_length = 1
    for i in range(n):
        w_i, h_i = pairs[i]
        for j in range(i):
            w_j, h_j = pairs[j]
            # Điều kiện LIS 2D: Cặp pairs[j] có thể đứng trước pairs[i]
            # nếu w_j < w_i VÀ h_j < h_i.
            # Do đã sắp xếp, điều kiện w_j <= w_i luôn đúng.
            # Ta chỉ cần kiểm tra điều kiện nghiêm ngặt w_j < w_i và h_j <
h_i.
            # Lưu ý: Nếu w_j = w_i, ta phải đảm bảo h_j < h_i.
            # Trong trường hợp sắp xếp theo (w tăng, h tăng), khi w_j = w_i,
            # j < i ngụ ý h_j <= h_i. Do đó, ta chỉ cần kiểm tra h_j < h_i
            # cùng với w_j < w_i để tránh các trường hợp bằng nhau.
            if w_j < w_i and h_j < h_i:
                # Cập nhật DP[i] dựa trên DP[j] lớn nhất có thể
                dp[i] = max(dp[i], dp[j] + 1)
        max_chain_length = max(max_chain_length, dp[i])
    return max_chain_length
# ---------------- CHẠY THỬ VÀ MINH HỌA ----------------
# Ví dụ 1:
# Chuỗi lớn nhất: (2, 3) -> (4, 5) -> (7, 8). Độ dài 3.
pairs1 = [(5, 4), (2, 3), (7, 8), (4, 5)]
```

```
print(f"Bo du lieu 1: {pairs1}")
result1 = longest_chain_of_pairs(pairs1)
print(f"Do dai chuoi lon nhat: {result1}") # Kết quả: 3
# Ví dụ 2:
# Chuỗi lớn nhất: (1, 2) -> (2, 3) -> (3, 4). Độ dài 3.
pairs2 = [(1, 2), (2, 3), (3, 4), (2, 1), (3, 2)]
print(f"\nBo du lieu 2: {pairs2}")
result2 = longest_chain_of_pairs(pairs2)
print(f"Do dai chuoi lon nhat: {result2}") # Kết quả: 3
# Ví dụ 3: Trường hợp bằng nhau (w=1, h=1) và (w=1, h=1)
# Sắp xếp: [(1, 1), (1, 1), (3, 3)]
# DP: dp[0]=1. dp[1]=1 (vì 1 không > 1). dp[2]=2 (từ (1,1) hoặc (1,1) -> (3,3)).
pairs3 = [(3, 3), (1, 1), (1, 1)]
print(f"\nBo du lieu 3: {pairs3}")
result3 = longest_chain_of_pairs(pairs3)
print(f"Do dai chuoi lon nhat: {result3}") # Kết quả: 2 (Chuỗi: (1, 1) -> (3, 3))
```

=> Kết quả sau khi chạy chương trình:

```
Bo du lieu 1: [(5, 4), (2, 3), (7, 8), (4, 5)]
Danh sach cac cap da sap xep: [(2, 3), (4, 5), (5, 4), (7, 8)]
Do dai chuoi lon nhat: 3

Bo du lieu 2: [(1, 2), (2, 3), (3, 4), (2, 1), (3, 2)]
Danh sach cac cap da sap xep: [(1, 2), (2, 1), (2, 3), (3, 2), (3, 4)]
Do dai chuoi lon nhat: 3

Bo du lieu 3: [(3, 3), (1, 1), (1, 1)]
Danh sach cac cap da sap xep: [(1, 1), (1, 1), (3, 3)]
Do dai chuoi lon nhat: 2
```

2.2.2. Code C++:

```
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
using namespace std;         // Tiền tố std
int longest_chain_of_pairs(vector<pair<int, int>> pairs)
{
    // Tim do dai chuoi lon nhat cac cap (w, h) sao cho moi cap sau deu lon
    // hon ca chieu rong va chieu cao cua cap truoc do.
    // Do phuc tap: O(N log N) cho sap xep, O(N^2) cho DP -> Tong O(N^2).
    // Tham so:
    // pairs (vector<pair<int,int>>): Danh sach cac cap (chieu rong, chieu cao).
    // Tra ve:
    // int: Do dai chuoi lon nhat.
    int n = pairs.size();
    if (n == 0)
    {
        return 0;
    }
    // BUOC 1: Sap xep danh sach cac cap (Preprocessing)
    // Sap xep theo chieu rong (w) tang dan.
    // Neu w bang nhau, sap xep theo chieu cao (h) tang dan.
    // Viec nay dam bao khi ta xet cap i, cac cap j truoc do da duoc xu ly
    // va ta chi can quan tam den dieu kien h_j < h_i.
    sort(pairs.begin(), pairs.end());
    cout << "Danh sach cac cap da sap xep: ";
    for (auto &p : pairs)
    {
        cout << "(" << p.first << ", " << p.second << ") ";
```

```
    }
    cout << endl;
    // BUOC 2: Ap dung Quy hoach dong (DP O(N^2))
    // dp[i] la do dai chuoi lon nhat ket thuc tai pairs[i].
    vector<int> dp(n, 1);
    int max_chain_length = 1;
    for (int i = 0; i < n; i++)
    {
        int w_i = pairs[i].first;
        int h_i = pairs[i].second;
        for (int j = 0; j < i; j++)
        {
            int w_j = pairs[j].first;
            int h_j = pairs[j].second;
            // Dieu kien LIS 2D: pairs[j] co the dung truoc pairs[i]
            // neu w_j < w_i VA h_j < h_i.
            // Do da sap xep, dieu kien w_j <= w_i luon dung.
            // Ta can kiem tra nghiem ngat w_j < w_i va h_j < h_i.
            // Neu w_j = w_i thi h_j phai < h_i de tranh trung lap.
            if (w_j < w_i && h_j < h_i)
            {
                // Cap nhat DP[i] dua tren DP[j] lon nhat co the
                dp[i] = max(dp[i], dp[j] + 1);
            }
        }
        max_chain_length = max(max_chain_length, dp[i]);
    }
    return max_chain_length;
}
// ----------------- CHAY THU VA MINH HOA -----------------
int main()
{
    // Vi du 1:
    // Chuoi lon nhat: (2,3) -> (4,5) -> (7,8). Do dai 3.
    vector<pair<int,int>> pairs1 = {{5,4}, {2,3}, {7,8}, {4,5}};
    cout << "Bo du lieu 1: ";
    for (auto &p : pairs1)
    {
        cout << "(" << p.first << ", " << p.second << ") ";
    }
    cout << endl;
    int result1 = longest_chain_of_pairs(pairs1);
    cout << "Do dai chuoi lon nhat: " << result1 << endl;
    // Vi du 2:
    // Chuoi lon nhat: (1,2) -> (2,3) -> (3,4). Do dai 3.
    vector<pair<int,int>> pairs2 = {{1,2}, {2,3}, {3,4}, {2,1}, {3,2}};
    cout << "\nBo du lieu 2: ";
    for (auto &p : pairs2)
    {
        cout << "(" << p.first << ", " << p.second << ") ";
    }
    cout << endl;
    int result2 = longest_chain_of_pairs(pairs2);
    cout << "Do dai chuoi lon nhat: " << result2 << endl;
    // Vi du 3: Truong hop bang nhau
    // Sap xep: (1,1), (1,1), (3,3)
    // Chuoi: (1,1) -> (3,3). Do dai 2.
    vector<pair<int,int>> pairs3 = {{3,3}, {1,1}, {1,1}};
```

```
    cout << "\nBo du lieu 3: ";
    for (auto &p : pairs3)
    {
        cout << "(" << p.first << ", " << p.second << ") ";
    }
    cout << endl;
    int result3 = longest_chain_of_pairs(pairs3);
    cout << "Do dai chuoi lon nhat: " << result3 << endl;
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
Bo du lieu 1: (5, 4) (2, 3) (7, 8) (4, 5)
Danh sach cac cap da sap xep: (2, 3) (4, 5) (5, 4) (7, 8)
Do dai chuoi lon nhat: 3

Bo du lieu 2: (1, 2) (2, 3) (3, 4) (2, 1) (3, 2)
Danh sach cac cap da sap xep: (1, 2) (2, 1) (2, 3) (3, 2) (3, 4)
Do dai chuoi lon nhat: 3

Bo du lieu 3: (3, 3) (1, 1) (1, 1)
Danh sach cac cap da sap xep: (1, 1) (1, 1) (3, 3)
Do dai chuoi lon nhat: 2
```

BÀI 27. GIỚI THIỆU BÀI TOÁN DP DẠNG CHUỖI – DẠNG BẢNG 2 CHIỀU

I. Mục Tiêu:

- **Kiến thức:** HS hiểu khái niệm **Dãy con chung dài nhất (LCS)**, xác định được các trường hợp trong công thức truy hồi và nhận diện đây là bài toán DP 2 chiều (2D DP).
- **Kỹ năng:** HS cài đặt được thuật toán LCS DP O(M . N) trong Python và C++ để tìm **độ dài** của LCS.
- **Tư duy:** HS phân tích được cấu trúc tối ưu của bài toán 2 dãy và cách xây dựng ma trận DP.

II. Phân Bổ Thời Gian (90 Phút):

Hoạt Động	Nội Dung	Thời gian (Phút)
Ôn tập & Giới thiệu	Định nghĩa LCS, so sánh với LIS, Cấu trúc DP 2D	15
Lý thuyết DP	Trạng thái, Công thức Truy hồi và 3 trường hợp	25
Thực hành	Cài đặt DP O(M . N) tìm độ dài (Python & C++)	40
Tổng kết	Cấu trúc ma trận, Bài tập về nhà	10
Tổng cộng		**90**

III. Lý Thuyết 1: Giới thiệu LCS (15 phút)

1. Dãy con chung dài nhất (LCS) là gì?

- **Dãy con (Subsequence):** Một dãy được tạo thành từ một dãy khác bằng cách xóa 0 hoặc nhiều phần tử mà không thay đổi thứ tự của các phần tử còn lại.
- **LCS:** Là dãy con dài nhất xuất hiện đồng thời trong cả hai dãy (hoặc chuỗi) đầu vào.
- **Ví dụ:**

  X = "AGGTAB"
  Y = "GXTXAYB"
  LCS là "GTAB" (Độ dài = 4).

2. Tại sao LCS là bài toán DP?

- **Cấu trúc tối ưu:** Lời giải LCS của hai chuỗi X và Y được xây dựng từ lời giải LCS của các tiền tố (prefix) của chúng.
- **Bài toán con chồng chéo:** Cùng một cặp tiền tố được đánh giá nhiều lần nếu sử dụng đệ quy thông thường.

IV. Lý thuyết 2: Công thức Truy hồi DP (25 phút)

- Ta định nghĩa ma trận DP hai chiều DP[i][j]:
  + **Trạng thái DP:** DP[i][j] là **độ dài của LCS** của hai tiền tố:
    X[1 … i] (tiền tố của chuỗi X có độ dài i).
    Y[1 … j] (tiền tố của chuỗi Y có độ dài j).
  + **Công thức Truy hồi (3 Trường hợp)**
    Xét ký tự cuối cùng của hai tiền tố: X[i] và Y[j].

Trường hợp	Điều kiện	Công thức	Ý nghĩa
1 (Match)	Nếu X[i] = Y[j]	DP[i][j] = DP[i-1][j-1] + 1	Ký tự này là một phần của LCS. Ta thêm 1 vào LCS của hai tiền tố ngắn hơn.
2 (Mismatch)	Nếu X[i] ≠ Y[j]	DP[i][j] = max(DP[i-1][j], DP[i][j-1])	Ký tự X[i] hoặc Y[j] không phải là một phần của LCS. Ta lấy giá trị tối đa từ việc loại bỏ một trong hai ký tự.
3 (Base Case)	i=0 hoặc j=0	DP[i][j] = 0	LCS của một chuỗi rỗng và chuỗi bất kỳ luôn là 0.

V. Thực Hành: Cài đặt DP O(M . N) (40 phút)

Bài toán: Cho hai chuỗi X và Y, tìm độ dài của LCS.

1. Code Python:

```
def longest_common_subsequence_length(X, Y):
    # Tìm độ dài của LCS giữa hai chuỗi X và Y bằng Quy hoạch động.
    # Độ phức tạp thời gian: O(M * N), trong đó M = len(X), N = len(Y).
    # Tham số:
    # X (str): Chuỗi thứ nhất.
    # Y (str): Chuỗi thứ hai.
    # Trả về:
    # int: Độ dài LCS.
    M = len(X)
    N = len(Y)
    # Ma trận DP (M+1 x N+1)
    # DP[i][j] lưu độ dài LCS của tiền tố X[0...i-1] và Y[0...j-1].
    # Kích thước M+1 và N+1 để xử lý Base Case (i=0 hoặc j=0) dễ dàng.
    # Khởi tạo ma trận với giá trị 0 (Base Case: LCS của chuỗi rỗng là 0).
    DP = [[0] * (N + 1) for _ in range(M + 1)]
    # Duyệt qua ma trận (DP O(M*N))
    for i in range(1, M + 1):
        for j in range(1, N + 1):
            # --- CÔNG THỨC TRUY HỒI ---
            # Trường hợp 1: Match (Ký tự cuối giống nhau)
            # Ký tự hiện tại là X[i-1] và Y[j-1] (chỉ mục 0-dựa)
            if X[i-1] == Y[j-1]:
                # LCS tăng lên 1 đơn vị so với LCS của hai tiền tố ngắn hơn (X[i-
2], Y[j-2])
                DP[i][j] = DP[i-1][j-1] + 1
            # Trường hợp 2: Mismatch (Ký tự cuối khác nhau)
            else:
                # LCS bằng giá trị lớn nhất từ việc loại bỏ X[i] (DP[i-1][j])
                # hoặc loại bỏ Y[j] (DP[i][j-1]).
                DP[i][j] = max(DP[i-1][j], DP[i][j-1])
    # Kết quả cuối cùng là giá trị ở góc dưới-phải của ma trận DP[M][N]
    return DP[M][N]
# ----------------- CHẠY THỬ VÀ MINH HỌA -----------------
X1 = "AGGTAB"
Y1 = "GXTXAYB"
# LCS là "GTAB", độ dài 4.
print(f"Chuoi X: {X1}")
print(f"Chuoi Y: {Y1}")
print(f"Do dai LCS (Python): {longest_common_subsequence_length(X1, Y1)}") # Kết
quả: 4
X2 = "ABCBDAB"
Y2 = "BDCABA"
# LCS là "BCBA" hoặc "BCDA" hoặc "BCABA". Độ dài 4.
print(f"\nChuoi X: {X2}")
print(f"Chuoi Y: {Y2}")
print(f"Do dai LCS (Python): {longest_common_subsequence_length(X2, Y2)}") # Kết
quả: 4
```

=> Kết quả sau khi chạy chương trình:

```
Chuoi X: AGGTAB
Chuoi Y: GXTXAYB
Do dai LCS (Python): 4

Chuoi X: ABCBDAB
Chuoi Y: BDCABA
Do dai LCS (Python): 4
```

- Giải thích Code Python:

Kích thước Ma trận: Sử dụng M+1 x N+1 để DP}[0][j] và DP}[i][0] tự động là Base Case (0). Khi truy cập ký tự chuỗi, ta dùng chỉ mục 0-dựa, ví dụ: X[i-1] tương ứng với ký tự thứ i của chuỗi X.

2. Code C++:

```
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
using namespace std;         // Tiền tố std
int longest_common_subsequence_length(string X, string Y)
{
    // Tim do dai cua LCS giua hai chuoi X va Y bang Quy hoach dong.
    // Do phuc tap thoi gian: O(M * N), trong do M = len(X), N = len(Y).
    // Tham so:
    // X (string): Chuoi thu nhat.
    // Y (string): Chuoi thu hai.
    // Tra ve:
    // int: Do dai LCS.
    int M = X.length();
    int N = Y.length();
    // Ma tran DP (M+1 x N+1)
    // DP[i][j] luu do dai LCS cua tien to X[0...i-1] va Y[0...j-1].
    // Kich thuoc M+1 va N+1 de xu ly Base Case (i=0 hoac j=0) de dang.
    // Khoi tao ma tran voi gia tri 0 (Base Case: LCS cua chuoi rong la 0).
    vector<vector<int>> DP(M + 1, vector<int>(N + 1, 0));
    // Duyet qua ma tran (DP O(M*N))
    for (int i = 1; i <= M; i++)
    {
        for (int j = 1; j <= N; j++)
        {
            // --- CONG THUC TRUY HOI ---
            // Truong hop 1: Match (Ky tu cuoi giong nhau)
            // Ky tu hien tai la X[i-1] va Y[j-1] (chi muc 0-dua)
            if (X[i - 1] == Y[j - 1])
            {
                // LCS tang len 1 don vi so voi LCS cua hai tien to ngan hon
                DP[i][j] = DP[i - 1][j - 1] + 1;
            }
            // Truong hop 2: Mismatch (Ky tu cuoi khac nhau)
            else
            {
                // LCS bang gia tri lon nhat tu viec loai bo X hoac Y
                DP[i][j] = max(DP[i - 1][j], DP[i][j - 1]);
            }
        }
    }
    // Ket qua cuoi cung nam o goc duoi-phai cua ma tran DP[M][N]
    return DP[M][N];
}
// ----------------- CHAY THU VA MINH HOA -----------------
int main()
{
    string X1 = "AGGTAB";
    string Y1 = "GXTXAYB";
    // LCS la "GTAB", do dai 4.
    cout << "Chuoi X: " << X1 << endl;
    cout << "Chuoi Y: " << Y1 << endl;
```

```
    cout << "Do dai LCS (C++): "
         << longest_common_subsequence_length(X1, Y1) << endl;
    string X2 = "ABCBDAB";
    string Y2 = "BDCABA";
    // LCS la "BCBA" hoac "BCDA" hoac "BCABA". Do dai 4.
    cout << "\nChuoi X: " << X2 << endl;
    cout << "Chuoi Y: " << Y2 << endl;
    cout << "Do dai LCS (C++): "
         << longest_common_subsequence_length(X2, Y2) << endl;
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
Chuoi X: AGGTAB
Chuoi Y: GXTXAYB
Do dai LCS (C++): 4

Chuoi X: ABCBDAB
Chuoi Y: BDCABA
Do dai LCS (C++): 4
```

- Giải thích Code C++:

Sử dụng vector<vector<int>>: C++ sử dụng vector lồng nhau để tạo ma trận DP. Lệnh **vector<vector<int>>(M + 1, vector<int>(N + 1, 0));** giúp khởi tạo ma trận **M+1 x N+1** với tất cả giá trị ban đầu là **0**.

VI. Tổng kết & Bài tập Về nhà (10 phút)

1. Tổng kết:

- LCS là bài toán DP 2 chiều với độ phức tạp O(M . N), nơi M và N là độ dài của hai chuỗi.
- Trạng thái DP[i][j] lưu trữ LCS của hai tiền tố.
- Vòng lặp bắt đầu từ i=1, j=1 để tận dụng Base Case DP[i][0] = 0 và DP[0][j] = 0.

2. Bài tập Về nhà:

2.1. Bài tập 1 (Xây dựng LCS): Nghiên cứu và cài đặt cách **truy vết** (backtrack) ma trận DP để in ra *một trong các* dãy con chung dài nhất, chứ không chỉ độ dài. (Gợi ý: Bắt đầu từ DP[M][N] và đi ngược lại).

2.1.1. Code Python:

```
def construct_longest_common_subsequence(X, Y):
    # Tìm độ dài và xây dựng lại một trong các Dãy con chung Dài nhất (LCS).
    # Sử dụng DP O(M * N) và Truy vết.
    # Tham số:
    # X (str): Chuỗi thứ nhất.
    # Y (str): Chuỗi thứ hai.
    # Trả về:
    # str: Chuỗi LCS.
    M = len(X)
    N = len(Y)
    # BƯỚC 1: Xây dựng Ma trận DP để tính độ dài (O(M*N))
    DP = [[0] * (N + 1) for _ in range(M + 1)]
    for i in range(1, M + 1):
        for j in range(1, N + 1):
            # Match
            if X[i-1] == Y[j-1]:
                DP[i][j] = DP[i-1][j-1] + 1
```

```
            # Mismatch
            else:
                DP[i][j] = max(DP[i-1][j], DP[i][j-1])
    # BƯỚC 2: Truy vết (Backtracking) để xây dựng chuỗi LCS (O(M+N))
    # Bắt đầu từ góc dưới-phải
    i = M
    j = N
    lcs_sequence = []
    while i > 0 and j > 0:
        # 1. Trường hợp Match (Ký tự giống nhau):
        # Nếu giá trị DP được lấy từ ô chéo trên (DP[i-1][j-1] + 1)
        if X[i-1] == Y[j-1]:
            lcs_sequence.append(X[i-1]) # Ký tự này là phần của LCS
            i -= 1                      # Di chuyển chéo lên
            j -= 1
        # 2. Trường hợp Mismatch (Không Match):
        # Di chuyển theo hướng có giá trị lớn hơn (hướng LCS tối ưu đến)
        elif DP[i-1][j] > DP[i][j-1]:
            # LCS đến từ việc loại bỏ X[i-1], đi lên
            i -= 1
        else:
            # LCS đến từ việc loại bỏ Y[j-1] (hoặc giá trị bằng nhau), đi sang trái
            j -= 1
    # Chuỗi được xây dựng ngược, nên cần đảo ngược lại
    return "".join(reversed(lcs_sequence))
# ----------------- CHẠY THỬ VÀ MINH HỌA -----------------
X1 = "AGGTAB"
Y1 = "GXTXAYB"
# LCS: "GTAB"
lcs_result_1 = construct_longest_common_subsequence(X1, Y1)
print(f"Chuoi X: {X1}")
print(f"Chuoi Y: {Y1}")
print(f"LCS: {lcs_result_1} (Do dai: {len(lcs_result_1)})")
X2 = "ABCBDAB"
Y2 = "BDCABA"
# LCS: "BCBA" (hoặc "BDAB")
lcs_result_2 = construct_longest_common_subsequence(X2, Y2)
print(f"\nChuoi X: {X2}")
print(f"Chuoi Y: {Y2}")
print(f"LCS: {lcs_result_2} (Do dai: {len(lcs_result_2)})")
```

=> Kết quả sau khi chạy chương trình:

```
Chuoi X: AGGTAB
Chuoi Y: GXTXAYB
LCS: GTAB (Do dai: 4)

Chuoi X: ABCBDAB
Chuoi Y: BDCABA
LCS: BDAB (Do dai: 4)
```

2.1.2. Code C++:

```
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
using namespace std;         // Tiền tố std
string construct_longest_common_subsequence(string X, string Y)
{
    // Tim do dai va xay dung lai mot trong cac Day con chung Dai nhat (LCS).
    // Su dung DP O(M * N) va Truy vet.
    // Tham so:
```

```
    // X (string): Chuoi thu nhat.
    // Y (string): Chuoi thu hai.
    // Tra ve:
    // string: Chuoi LCS.
    int M = X.length();
    int N = Y.length();
    // BUOC 1: Xay dung Ma tran DP de tinh do dai (O(M*N))
    vector<vector<int>> DP(M + 1, vector<int>(N + 1, 0));
    for (int i = 1; i <= M; i++)
    {
        for (int j = 1; j <= N; j++)
        {
            // Match
            if (X[i - 1] == Y[j - 1])
            {
                DP[i][j] = DP[i - 1][j - 1] + 1;
            }
            // Mismatch
            else
            {
                DP[i][j] = max(DP[i - 1][j], DP[i][j - 1]);
            }
        }
    }
    // BUOC 2: Truy vet (Backtracking) de xay dung chuoi LCS (O(M+N))
    // Bat dau tu goc duoi-phai
    int i = M;
    int j = N;
    vector<char> lcs_sequence;
    while (i > 0 && j > 0)
    {
        // 1. Truong hop Match (Ky tu giong nhau)
        // Neu gia tri DP duoc lay tu o cheo tren
        if (X[i - 1] == Y[j - 1])
        {
            lcs_sequence.push_back(X[i - 1]); // Ky tu nay la phan cua LCS
            i--;                              // Di chuyen cheo len
            j--;
        }
        // 2. Truong hop Mismatch (Khong Match)
        // Di chuyen theo huong co gia tri lon hon
        else if (DP[i - 1][j] > DP[i][j - 1])
        {
            // LCS den tu viec loai bo X[i-1], di len
            i--;
        }
        else
        {
            // LCS den tu viec loai bo Y[j-1], di sang trai
            j--;
        }
    }
    // Chuoi duoc xay dung nguoc, nen can dao nguoc lai
    reverse(lcs_sequence.begin(), lcs_sequence.end());
    // Chuyen vector ky tu thanh chuoi
    return string(lcs_sequence.begin(), lcs_sequence.end());
}
// ----------------- CHAY THU VA MINH HOA -----------------
```

```
int main()
{
    string X1 = "AGGTAB";
    string Y1 = "GXTXAYB";
    // LCS: "GTAB"
    string lcs_result_1 = construct_longest_common_subsequence(X1, Y1);
    cout << "Chuoi X: " << X1 << endl;
    cout << "Chuoi Y: " << Y1 << endl;
    cout << "LCS: " << lcs_result_1
         << " (Do dai: " << lcs_result_1.length() << ")" << endl;
    string X2 = "ABCBDAB";
    string Y2 = "BDCABA";
    // LCS: "BCBA" hoac "BDAB"
    string lcs_result_2 = construct_longest_common_subsequence(X2, Y2);
    cout << "\nChuoi X: " << X2 << endl;
    cout << "Chuoi Y: " << Y2 << endl;
    cout << "LCS: " << lcs_result_2
         << " (Do dai: " << lcs_result_2.length() << ")" << endl;
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
Chuoi X: AGGTAB
Chuoi Y: GXTXAYB
LCS: GTAB (Do dai: 4)

Chuoi X: ABCBDAB
Chuoi Y: BDCABA
LCS: BDAB (Do dai: 4)
```

2.2. Bài tập 2 (Ứng dụng): Cho hai chuỗi X và Y, tìm số lượng các ký tự cần chèn và xóa tối thiểu để biến X thành Y. (Gợi ý: Bài toán này liên quan chặt chẽ đến LCS).

2.2.1. Code Python:

```
def longest_common_subsequence_length(X, Y):
    # Tìm độ dài của LCS giữa hai chuỗi X và Y bằng Quy hoạch động (O(M*N)).
    M = len(X)
    N = len(Y)
    DP = [[0] * (N + 1) for _ in range(M + 1)]
    for i in range(1, M + 1):
        for j in range(1, N + 1):
            if X[i-1] == Y[j-1]:
                DP[i][j] = DP[i-1][j-1] + 1
            else:
                DP[i][j] = max(DP[i-1][j], DP[i][j-1])
    return DP[M][N]
def min_deletions_and_insertions(X, Y):
    # Tìm số lượng thao tác xóa và chèn tối thiểu để biến X thành Y.
    # Tham số:
    # X (str): Chuỗi nguồn.
    # Y (str): Chuỗi đích.
    # Trả về:
    # tuple: (min_deletions, min_insertions, total_operations)
    # 1. Tính độ dài LCS giữa X và Y
    lcs_len = longest_common_subsequence_length(X, Y)
    # Độ dài chuỗi nguồn và chuỗi đích
    M = len(X)
    N = len(Y)
    # 2. Tính số lần XÓA tối thiểu (Số ký tự trong X không thuộc LCS)
```

```
    min_deletions = M - lcs_len
    # 3. Tính số lần CHÈN tối thiểu (Số ký tự trong Y không thuộc LCS)
    min_insertions = N - lcs_len
    # 4. Tổng thao tác tối thiểu
    total_operations = min_deletions + min_insertions
    return (min_deletions, min_insertions, total_operations)
# ----------------- CHẠY THỬ VÀ MINH HỌA -----------------
X1 = "HEAP"
Y1 = "PEA"
# LCS("HEAP", "PEA") = "EA" (Độ dài = 2)
# Xóa: X = HEAP (4 ký tự). Giữ lại "EA" (2 ký tự) -> Phải xóa 'H' và 'P'. Xóa = 4 -
2 = 2.
# Chèn: Y = PEA (3 ký tự). Giữ lại "EA" (2 ký tự) -> Phải chèn 'P'. Chèn = 3 - 2 =
1.
# Tổng thao tác: 2 Xóa + 1 Chèn = 3.
min_del, min_ins, total = min_deletions_and_insertions(X1, Y1)
print(f"Chuoi X: '{X1}', Chuoi Y: '{Y1}'")
print(f"LCS Length: {len(X1) + len(Y1) - total}")
print(f"So lan XOA toi thieu (X -> Y): {min_del}") # 2
print(f"So lan CHEN toi thieu (X -> Y): {min_ins}") # 1
print(f"Tong thao tac toi thieu: {total}") # 3
# Ví dụ 2: X = "GXTXAYB", Y = "AGGTAB"
X2 = "GXTXAYB"
Y2 = "AGGTAB"
# LCS("GXTXAYB", "AGGTAB") = "GTAB" (Độ dài = 4)
# Xóa: 7 - 4 = 3
# Chèn: 6 - 4 = 2
min_del, min_ins, total = min_deletions_and_insertions(X2, Y2)
print(f"\nChuoi X: '{X2}', Chuoi Y: '{Y2}'")
print(f"So lan XOA toi thieu (X -> Y): {min_del}") # 3
print(f"So lan CHEN toi thieu (X -> Y): {min_ins}") # 2
print(f"Tong thao tac toi thieu: {total}") # 5
```

=> Kết quả sau khi chạy chương trình:

```
Chuoi X: 'HEAP', Chuoi Y: 'PEA'
LCS Length: 4
So lan XOA toi thieu (X -> Y): 2
So lan CHEN toi thieu (X -> Y): 1
Tong thao tac toi thieu: 3

Chuoi X: 'GXTXAYB', Chuoi Y: 'AGGTAB'
So lan XOA toi thieu (X -> Y): 3
So lan CHEN toi thieu (X -> Y): 2
Tong thao tac toi thieu: 5
```

2.2.2. Code C++:

```
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
using namespace std;         // Tiền tố std
int longest_common_subsequence_length(string X, string Y)
{
    // Tim do dai cua LCS giua hai chuoi X va Y bang Quy hoach dong (O(M*N)).
    int M = X.length();
    int N = Y.length();
    // Ma tran DP kich thuoc (M+1) x (N+1)
    vector<vector<int>> DP(M + 1, vector<int>(N + 1, 0));
    for (int i = 1; i <= M; i++)
    {
        for (int j = 1; j <= N; j++)
```

```
        {
            if (X[i - 1] == Y[j - 1])
            {
                DP[i][j] = DP[i - 1][j - 1] + 1;
            }
            else
            {
                DP[i][j] = max(DP[i - 1][j], DP[i][j - 1]);
            }
        }
    }
    return DP[M][N];
}
tuple<int, int, int> min_deletions_and_insertions(string X, string Y)
{
    // Tim so luong thao tac xoa va chen toi thieu de bien X thanh Y.
    // Tham so:
    // X (string): Chuoi nguon.
    // Y (string): Chuoi dich.
    // Tra ve:
    // tuple: (min_deletions, min_insertions, total_operations)
    // 1. Tinh do dai LCS giua X va Y
    int lcs_len = longest_common_subsequence_length(X, Y);
    // Do dai chuoi nguon va chuoi dich
    int M = X.length();
    int N = Y.length();
    // 2. Tinh so lan XOA toi thieu
    int min_deletions = M - lcs_len;
    // 3. Tinh so lan CHEN toi thieu
    int min_insertions = N - lcs_len;
    // 4. Tong thao tac toi thieu
    int total_operations = min_deletions + min_insertions;
    return make_tuple(min_deletions, min_insertions, total_operations);
}
// ----------------- CHAY THU VA MINH HOA -----------------
int main()
{
    string X1 = "HEAP";
    string Y1 = "PEA";
    // LCS("HEAP", "PEA") = "EA" (Do dai = 2)
    // Xoa = 4 - 2 = 2
    // Chen = 3 - 2 = 1
    // Tong = 3
    int min_del, min_ins, total;
    tie(min_del, min_ins, total) = min_deletions_and_insertions(X1, Y1);
    cout << "Chuoi X: '" << X1 << "', Chuoi Y: '" << Y1 << "'" << endl;
    cout << "So lan XOA toi thieu (X -> Y): " << min_del << endl;
    cout << "So lan CHEN toi thieu (X -> Y): " << min_ins << endl;
    cout << "Tong thao tac toi thieu: " << total << endl;
    // Vi du 2
    string X2 = "GXTXAYB"; string Y2 = "AGGTAB";
    tie(min_del, min_ins, total) = min_deletions_and_insertions(X2, Y2);
    cout << "\nChuoi X: '" << X2 << "', Chuoi Y: '" << Y2 << "'" << endl;
    cout << "So lan XOA toi thieu (X -> Y): " << min_del << endl;
    cout << "So lan CHEN toi thieu (X -> Y): " << min_ins << endl;
    cout << "Tong thao tac toi thieu: " << total << endl;
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
Chuoi X: 'HEAP', Chuoi Y: 'PEA'
LCS Length: 4
So lan XOA toi thieu (X -> Y): 2
So lan CHEN toi thieu (X -> Y): 1
Tong thao tac toi thieu: 3

Chuoi X: 'GXTXAYB', Chuoi Y: 'AGGTAB'
So lan XOA toi thieu (X -> Y): 3
So lan CHEN toi thieu (X -> Y): 2
Tong thao tac toi thieu: 5
```

CHỦ ĐỀ 10. THUẬT TOÁN QUAY LUI

BÀI 28. SINH TỔ HỢP

I. Mục Tiêu:

- **Kiến thức:** HS hiểu rõ nguyên lý của thuật toán Quay lui (Backtracking), đặc biệt là cách áp dụng nó để giải quyết các bài toán tìm kiếm và liệt kê (như Sinh Tổ hợp).
- **Kỹ năng:** HS cài đặt được thuật toán quay lui để liệt kê tổ hợp chập **K** của **N** phần tử trong Python và C++.
- **Tư duy:** HS phân tích được cây tìm kiếm (Search Tree), xác định các điều kiện dừng (Base Case) và điều kiện loại bỏ (Pruning) để tối ưu hóa quá trình tìm kiếm.

II. Phân Bổ Thời Gian (90 Phút):

Hoạt Động	Nội Dung	Thời gian (Phút)
Ôn tập & Giới thiệu	Định nghĩa Quay lui (Backtracking), Cây tìm kiếm và Sinh Tổ hợp	15
Lý thuyết	Công thức Truy hồi và Nguyên tắc Lựa chọn	25
Thực hành	Bài 1. Cài đặt Sinh Tổ hợp Chập K (Python & C++)	40
Tổng kết	So sánh với Hoán vị, Bài tập về nhà	10
Tổng cộng		**90**

III. Lý Thuyết 1: Giới thiệu Quay lui & Tổ hợp (15 phút)

1. Thuật toán Quay lui (Backtracking):

Quay lui là một kỹ thuật tìm kiếm theo chiều sâu (Depth-First Search - DFS) được sử dụng để tìm tất cả các giải pháp cho một bài toán.

- **Nguyên tắc:** Xây dựng giải pháp từng bước một.
  + Tại mỗi bước, ta cố gắng thêm một thành phần vào giải pháp hiện tại.

+ Nếu việc thêm thành phần đó dẫn đến một tình trạng không thể là giải pháp hợp lệ (**điều kiện loại bỏ**), ta **quay lui** (bỏ thành phần đó và thử lựa chọn khác).

+ Nếu tìm được giải pháp đầy đủ (**điều kiện dừng**), ta ghi nhận và tiếp tục quay lui để tìm giải pháp khác.

2. Bài toán Sinh Tổ hợp (Combinations):

- Cho tập hợp S = {1, 2, …, N}, tìm tất cả các tập hợp con có **K** phần tử.
- **Điều kiện Tổ hợp:** Thứ tự các phần tử không quan trọng (tức là {1, 3} giống {3, 1}).

3. Cấu trúc Quay lui cho Tổ hợp:

- Ta cần định nghĩa hàm quay lui **Try(k, start_val)**:
  + **k**: Vị trí hiện tại ta đang chọn phần tử cho tổ hợp (từ 1 đến K).
  + **start_val**: Giá trị nhỏ nhất có thể chọn cho vị trí hiện tại.
- Để đảm bảo không có sự lặp lại và tuân thủ thứ tự không giảm dần (tính chất Tổ hợp), ta chọn phần tử **A[k]** từ **start_val** đến **N**.

IV. Lý thuyết 2: Công thức Truy hồi và Ràng buộc (25 phút)

1. Mảng kết quả và Biến trạng thái:

- **Mảng A:** Mảng kích thước **K** để lưu trữ tổ hợp hiện tại.
- **Hàm đệ quy: solve(index, start_value)**

2. Công thức Truy hồi:

- **Điều kiện Dừng (Base Case):** Nếu đã chọn đủ K phần tử (index = K), in ra kết quả.
- **Lựa chọn và Đệ quy:** Ta thử các giá trị **i** cho phần tử **A[index]**:
  + Cho **i** chạy từ **start_value** đến **N - K + index**
  + **Ràng buộc: N - K + index** là giới hạn trên đảm bảo luôn có đủ **K - index** phần tử còn lại để hoàn thành tổ hợp.
  + Đặt **A[index] = i**.
  + Gọi đệ quy: **solve(index + 1, i + 1)**. (Phần tử tiếp theo phải lớn hơn **i**).
- **Quay lui (Backtrack):** Sau khi gọi đệ quy, ta không cần phải hủy bỏ (undo) các thay đổi vì mảng **A** được xử lý theo chỉ mục.

V. Thực Hành: Cài đặt Sinh Tổ hợp (40 phút)

- **Bài toán:** Cho **N** và **K**, liệt kê tất cả các tổ hợp chập **K** của **{1, 2, …, N}.**

1. Code Python:

```
def combinations_backtracking(N, K):
    # Sử dụng thuật toán Quay lui (Backtracking) để liệt kê tất cả
    # các tổ hợp chập K của tập {1, 2, ..., N}.
    # Độ phức tạp: O(C(N, K) * K) (Số lần in kết quả nhân với thời gian in)
    # Tham số:
    # N (int): Số phần tử của tập hợp gốc (từ 1 đến N).
    # K (int): Số phần tử trong mỗi tổ hợp (chập K).
    if K > N or K < 0:
        print("K phải nằm trong khoảng [0, N]")
        return
    # Mảng để lưu trữ tổ hợp hiện tại. Kích thước K.
    current_combination = [0] * K
    def solve(k, start_val):
        # Hàm quay lui (DFS) để chọn giá trị cho vị trí k (chỉ mục 0-dựa).
```

```
        # Tham số:
        # k (int): Chỉ mục của phần tử đang được chọn trong tổ hợp (từ 0 đến K-1).
        # start_val (int): Giá trị nhỏ nhất có thể chọn cho vị trí k.
        #                  (Đảm bảo tính chất Tổ hợp: A[k] > A[k-1]).
        # 1. ĐIỀU KIỆN DỪNG (BASE CASE)
        if k == K:
            # Đã chọn đủ K phần tử, in ra kết quả
            print(current_combination)
            return
        # 2. ĐIỀU KIỆN CẮT TỈA (PRUNING) - Giới hạn trên
        # Đảm bảo rằng từ vị trí k, luôn còn đủ (K - k) phần tử để chọn
        # Giới hạn max_val: N - (K - k) + 1 + 1 (Do phần tử đầu tiên của mảng bắt
đầu từ 0 nên phải cộng thêm 1)
        max_val = N - K + k + 1 + 1
        # 3. VÒNG LẶP (LỰA CHỌN)
        for i in range(start_val, max_val):
            # Đặt giá trị i vào vị trí k
            current_combination[k] = i
            # 4. ĐỆ QUY (CHUYỂN TRẠNG THÁI)
            # Chuyển sang vị trí tiếp theo (k + 1)
            # Giá trị bắt đầu mới: i + 1 (Bắt buộc lớn hơn giá trị vừa chọn)
            solve(k + 1, i + 1)
            # 5. QUAY LUI (BACKTRACK)
            # Trong Python, không cần thao tác undo tường minh vì giá trị
            # current_combination[k] sẽ bị ghi đè khi vòng lặp chuyển sang i tiếp
theo.
    print(f"--- To hop chap {K} cua {N} (Tap {{1..{N}}}) ---")
    # Bắt đầu chọn phần tử đầu tiên (vị trí k=0), giá trị bắt đầu là 1
    solve(0, 1)
# ---------------- CHẠY THỬ VÀ MINH HỌA ----------------
N = 4
K = 2
# Kết quả mong đợi: [1, 2], [1, 3], [1, 4], [2, 3], [2, 4], [3, 4]
combinations_backtracking(N, K)
```

=> Kết quả sau khi chạy chương trình:

```
--- To hop chap 2 cua 4 (Tap {1..4}) ---
[1, 2]
[1, 3]
[1, 4]
[2, 3]
[2, 4]
[3, 4]
```

- Giải thích Code Python:

k và start_val: k là chỉ mục trong mảng A (0-dựa, từ 0 đến K-1). **start_val** đảm bảo giá trị tiếp theo lớn hơn giá trị vừa chọn, tránh lặp lại và đảm bảo tính chất tổ hợp (ví dụ: sau khi chọn 2, không chọn 1 nữa).

2. Code C++:

```
#include <iostream>         // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>    // Thư viện tổng hợp
using namespace std;        // Tiền tố std
void combinations_backtracking(int N, int K)
{
    // Su dung thuat toan Quay lui (Backtracking) de liet ke tat ca
    // cac to hop chap K cua tap {1, 2, ..., N}.
    // Do phuc tap: O(C(N, K) * K)
```

```
    if (K > N || K < 0)
    {
        cout << "K phai nam trong khoang [0, N]" << endl;
        return;
    }
    // Mang de luu tru to hop hien tai. Kich thuoc K.
    vector<int> current_combination(K);
    // Ham quay lui (DFS)
    // k: chi muc dang duoc chon (0-dựa)
    // start_val: gia tri nho nhat co the chon cho vi tri k
    function<void(int, int)> solve = [&](int k, int start_val)
    {
        // 1. DIEU KIEN DUNG (BASE CASE)
        if (k == K)
        {
            // Da chon du K phan tu, in ra ket qua
            cout << "[";
            for (int i = 0; i < K; i++)
            {
                cout << current_combination[i];
                if (i < K - 1) cout << ", ";
            }
            cout << "]" << endl;
            return;
        }
        // 2. DIEU KIEN CAT TIA (PRUNING)
        // Dam bao luon con du (K - k) phan tu de chon
        int max_val = N - K + k + 1 + 1;
        // 3. VONG LAP (LUA CHON)
        for (int i = start_val; i < max_val; i++)
        {
            // Dat gia tri i vao vi tri k
            current_combination[k] = i;
            // 4. DE QUY (CHUYEN TRANG THAI)
            // Chon phan tu tiep theo, bat dau tu i + 1
            solve(k + 1, i + 1);
            // 5. QUAY LUI (BACKTRACK)
            // Khong can thao tac undo tuong minh
            // Vi current_combination[k] se bi ghi de o vong lap tiep theo
        }
    };
    cout << "--- To hop chap " << K << " cua " << N << " (Tap {1.." << N << "}) ---
" << endl;
    // Bat dau chon phan tu dau tien (k = 0), gia tri bat dau la 1
    solve(0, 1);
}
// ---------------- CHAY THU VA MINH HOA ----------------
int main()
{
    int N = 4;
    int K = 2;
    // Ket qua mong doi:
    // [1, 2], [1, 3], [1, 4], [2, 3], [2, 4], [3, 4]
    combinations_backtracking(N, K);
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
--- To hop chap 2 cua 4 (Tap {1..4}) ---
[1, 2]
[1, 3]
[1, 4]
[2, 3]
[2, 4]
[3, 4]
```

VI. Tổng kết & Bài tập Về nhà (10 phút):

1. Tóm tắt Nguyên lý Quay lui cho Tổ hợp:

- **Quản lý Vị trí và Giá trị:** Sử dụng hai biến: **index** (vị trí đang chọn) và **start_value** (giá trị tối thiểu có thể chọn).
- **Cắt tỉa (Pruning):** Ràng buộc **max_val (N - K + k + 1) + 1** đảm bảo vòng lặp không cố gắng chọn một giá trị mà không đủ các phần tử còn lại để hoàn thành tổ hợp có độ dài **K**.
- **Đảm bảo tính chất Tổ hợp:** Luôn gọi đệ quy với **start_value** là **giá trị vừa chọn cộng 1 (i + 1)** để đảm bảo **A[k+1] > A[k]**, tránh lặp lại và tạo ra các hoán vị.

2. Bài tập Về nhà:

2.1. Bài tập 1 (Tổ hợp của Tập hợp bất kỳ): Mở rộng thuật toán để sinh tổ hợp chập K của một tập hợp S bất kỳ (không phải {1, …, N}). (Gợi ý: Sắp xếp S trước, sau đó truyền chỉ mục thay vì giá trị).

2.1.1. Code Python:

```python
def combinations_of_set(S, K):
    # Sinh tất cả các tổ hợp chập K của một tập hợp S bất kỳ.
    # Sử dụng thuật toán Quay lui dựa trên chỉ mục (index).
    # Tham số:
    # S (list): Tập hợp các phần tử bất kỳ.
    # K (int): Số phần tử trong mỗi tổ hợp.
    N = len(S)
    if K > N or K < 0:
        print("K phai nam trong khoang [0, len(S)]")
        return
    # BƯỚC 1: Sắp xếp tập hợp S để đảm bảo tính chất tổ hợp (tránh trùng lặp)
    # Nếu S là tập hợp các số, sắp xếp giúp đảm bảo A[i] < A[i+1].
    S.sort()
    current_combination = [None] * K
    all_combinations = []
    def solve(k, start_index):
        # Hàm quay lui dựa trên chỉ mục.
        # k (int): Chỉ mục của vị trí đang điền vào tổ hợp (0 đến K-1).
        # start_index (int): Chỉ mục đầu tiên trong S có thể chọn cho vị trí k.
        # ĐIỀU KIỆN DỪNG: Đã điền đủ K phần tử
        if k == K:
            # Thêm bản sao của tổ hợp hiện tại vào kết quả
            all_combinations.append(list(current_combination))
            return
        # ĐIỀU KIỆN CẮT TỈA (PRUNING) - Giới hạn trên
        # Đảm bảo còn đủ K - k phần tử còn lại trong S để hoàn thành tổ hợp
        # Giới hạn chỉ mục tối đa có thể chọn: N - (K - k)
        max_index = N - K + k
        # VÒNG LẶP (LỰA CHỌN)
        # i là chỉ mục trong mảng S. Vòng lặp chạy từ start_index đến max_index.
        for i in range(start_index, max_index + 1):
            # Đặt giá trị: Chọn phần tử S[i]
```

```
            current_combination[k] = S[i]
            # ĐỆ QUY (CHUYỂN TRẠNG THÁI)
            # Chuyển sang vị trí tiếp theo (k + 1)
            # Chỉ mục bắt đầu mới: i + 1 (để đảm bảo tính chất tổ hợp: chỉ chọn các
phần tử sau S[i])
            solve(k + 1, i + 1)
    print(f"--- To hop chap {K} cua tap {S} ---")
    # Bắt đầu chọn phần tử đầu tiên (k=0), chỉ mục bắt đầu từ 0
    solve(0, 0)
    return all_combinations
# ----------------- CHẠY THỬ VÀ MINH HỌA -----------------
# Tập hợp S bất kỳ (có cả số âm và thứ tự lộn xộn)
S1 = [3, -1, 4, 1, 0]
K1 = 3
# Sắp xếp S1: [-1, 0, 1, 3, 4]
# C(5, 3) = 10 tổ hợp.
result1 = combinations_of_set(S1, K1)
print(f"Tong so to hop tim duoc: {len(result1)}")
# print(result1)
# Ví dụ kết quả: [-1, 0, 1], [-1, 0, 3], [-1, 0, 4], ..., [1, 3, 4]
# Tập hợp S chứa các chuỗi (Strings)
S2 = ["a", "b", "c", "d"]
K2 = 2
# Sắp xếp S2: ["a", "b", "c", "d"]
# C(4, 2) = 6 tổ hợp.
result2 = combinations_of_set(S2, K2)
print(f"\nTong so to hop tim duoc: {len(result2)}")
print(result2)
# Kết quả: ['a', 'b'], ['a', 'c'], ['a', 'd'], ['b', 'c'], ['b', 'd'], ['c', 'd']
```

=> Kết quả sau khi chạy chương trình:

```
--- To hop chap 3 cua tap [-1, 0, 1, 3, 4] ---
Tong so to hop tim duoc: 10
--- To hop chap 2 cua tap ['a', 'b', 'c', 'd'] ---

Tong so to hop tim duoc: 6
[['a', 'b'], ['a', 'c'], ['a', 'd'], ['b', 'c'], ['b', 'd'], ['c', 'd']]
```

2.1.2. Code C++:

```
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
using namespace std;         // Tiền tố std
template <typename T>
vector<vector<T>> combinations_of_set(vector<T> S, int K)
{
    // Sinh tat ca cac to hop chap K cua mot tap hop S bat ky.
    // Su dung thuat toan Quay lui dua tren chi muc (index).
    // Tham so:
    // S (vector<T>): Tap hop cac phan tu bat ky.
    // K (int): So phan tu trong moi to hop.
    int N = S.size();
    if (K > N || K < 0)
    {
        cout << "K phai nam trong khoang [0, len(S)]" << endl;
        return {};
    }
    // BUOC 1: Sap xep tap hop S de dam bao tinh chat to hop (tranh trung lap)
    sort(S.begin(), S.end());
```

```
    vector<T> current_combination(K);
    vector<vector<T>> all_combinations;
    // Ham quay lui dua tren chi muc
    function<void(int, int)> solve = [&](int k, int start_index)
    {
        // DIEU KIEN DUNG: Da dien du K phan tu
        if (k == K)
        {
            // Them ban sao cua to hop hien tai vao ket qua
            all_combinations.push_back(current_combination);
            return;
        }
        // DIEU KIEN CAT TIA (PRUNING)
        // Dam bao con du K - k phan tu de hoan thanh to hop
        int max_index = N - K + k;
        // VONG LAP (LUA CHON)
        // i la chi muc trong mang S
        for (int i = start_index; i <= max_index; i++)
        {
            // Dat gia tri: Chon phan tu S[i]
            current_combination[k] = S[i];
            // DE QUY (CHUYEN TRANG THAI)
            // Chuyen sang vi tri tiep theo (k + 1)
            // Chi muc bat dau moi: i + 1
            solve(k + 1, i + 1);
        }
    };
    cout << "--- To hop chap " << K << " cua tap { ";
    for (const auto& x : S) cout << x << " ";
    cout << "} ---" << endl;
    // Bat dau chon phan tu dau tien (k = 0), chi muc bat dau tu 0
    solve(0, 0);
    return all_combinations;
}
int main()
{
    // Vi du 1: Tap hop so nguyen
    vector<int> S1 = {3, -1, 4, 1, 0};
    int K1 = 3;
    auto result1 = combinations_of_set(S1, K1);
    cout << "Tong so to hop tim duoc: " << result1.size() << endl;
    // Vi du 2: Tap hop chuoi
    vector<string> S2 = {"a", "b", "c", "d"};
    int K2 = 2;
    auto result2 = combinations_of_set(S2, K2);
    cout << "\nTong so to hop tim duoc: " << result2.size() << endl;
    for (const auto& comb : result2)
    {
        cout << "[ ";
        for (const auto& x : comb) cout << x << " ";
        cout << "]" << endl;
    }
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
--- To hop chap 3 cua tap { -1 0 1 3 4 } ---
Tong so to hop tim duoc: 10
--- To hop chap 2 cua tap { a b c d } ---

Tong so to hop tim duoc: 6
[ a b ]
[ a c ]
[ a d ]
[ b c ]
[ b d ]
[ c d ]
```

2.2. Bài tập 2 (Sinh Tổ hợp Tổng bằng X): Cho một tập hợp S và một số T, tìm tất cả các tổ hợp (dãy con) trong S có tổng bằng T. (Bài toán này còn gọi là *Combination Sum*).

2.2.1. Code Python:

```python
def combination_sum_unique(S, T):
    # Sử dụng thuật toán Quay lui (Backtracking) để tìm tất cả các tổ hợp
    # (dãy con) trong tập S có tổng bằng T.
    # Mỗi phần tử trong S được sử dụng tối đa một lần trong mỗi tổ hợp.
    # Tham số:
    # S (list): Tập hợp các số nguyên.
    # T (int): Tổng mục tiêu.
    # Trả về:
    # list of lists: Danh sách tất cả các tổ hợp hợp lệ.
    # Sắp xếp S: Giúp tối ưu quá trình quay lui (cắt tỉa nhánh) và
    # đảm bảo các tổ hợp được in ra theo thứ tự tăng dần.
    S.sort()
    N = len(S)
    all_combinations = []
    def solve(target, start_index, current_combination):
        # 1. ĐIỀU KIỆN DỪNG
        if target == 0:
            # Tìm thấy một tổ hợp có tổng bằng T. Ghi nhận kết quả.
            all_combinations.append(list(current_combination))
            return
        # Nếu target < 0, nhánh này không thể dẫn đến lời giải hợp lệ
        if target < 0:
            return
        # 2. VÒNG LẶP (LỰA CHỌN VÀ CẮT TỈA)
        # i là chỉ mục trong S
        for i in range(start_index, N):
            # Cắt tỉa nhánh 1 (Pruning): Nếu S[i] lớn hơn target còn lại,
            # các phần tử sau đó cũng sẽ lớn hơn (vì S đã sắp xếp), nên ta dừng
vòng lặp.
            if S[i] > target:
                break
            # Thêm S[i] vào tổ hợp hiện tại (Lựa chọn)
            current_combination.append(S[i])
            # ĐỆ QUY (Chuyển trạng thái)
            # target mới: target - S[i]
            # start_index mới: i + 1 (Đảm bảo S[i] không được chọn lại, giữ tính
duy nhất)
            solve(target - S[i], i + 1, current_combination)
            # QUAY LUI (BACKTRACK)
            # Loại bỏ S[i] khỏi tổ hợp để thử lựa chọn tiếp theo của vòng lặp
            current_combination.pop()
```

```
    print(f"--- Sinh To hop co Tong bang {T} tu tap {S} ---")
    # Bắt đầu với tổng T, chỉ mục 0, tổ hợp rỗng
    solve(T, 0, [])
    return all_combinations
# ---------------- CHẠY THỬ VÀ MINH HỌA ----------------
S1 = [10, 1, 2, 7, 6, 1, 5]
T1 = 8
# Sắp xếp: [1, 1, 2, 5, 6, 7, 10]
# Kết quả mong đợi: [1, 7], [1, 2, 5], [2, 6] (Chỉ sử dụng 1 lần, bỏ qua trùng lặp)
result1 = combination_sum_unique(S1, T1)
print(f"Ket qua tim duoc: {result1}")
# Ví dụ kết quả: [1, 7], [1, 2, 5], [2, 6] (Nếu xử lý giá trị 1 lặp lại)
# Ví dụ 2:
S2 = [2, 3, 5]
T2 = 8
# Kết quả mong đợi: [3, 5]
result2 = combination_sum_unique(S2, T2)
print(f"Ket qua tim duoc: {result2}")
```

=> Kết quả sau khi chạy chương trình:

```
--- Sinh To hop co Tong bang 8 tu tap [1, 1, 2, 5, 6, 7, 10] ---
Ket qua tim duoc: [[1, 1, 6], [1, 2, 5], [1, 7], [1, 2, 5], [1, 7], [2, 6]]
--- Sinh To hop co Tong bang 8 tu tap [2, 3, 5] ---
Ket qua tim duoc: [[3, 5]]
```

2.2.2. Code C++:

```
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
using namespace std;         // Tiền tố std
vector<vector<int>> combination_sum_unique(vector<int> S, int T)
{
    // Su dung thuat toan Quay lui (Backtracking) de tim tat ca cac to hop
    // (day con) trong tap S co tong bang T.
    // Moi phan tu trong S duoc su dung toi da mot lan trong moi to hop.
    // Tham so:
    // S (vector<int>): Tap hop cac so nguyen.
    // T (int): Tong muc tieu.
    // Tra ve:
    // vector<vector<int>>: Danh sach tat ca cac to hop hop le.
    // Sap xep S: Giup toi uu qua trinh quay lui (cat tia nhanh)
    // va dam bao cac to hop duoc in ra theo thu tu tang dan.
    sort(S.begin(), S.end());
    int N = S.size();
    vector<vector<int>> all_combinations;
    vector<int> current_combination;
    // Ham quay lui
    function<void(int, int)> solve = [&](int target, int start_index)
    {
        // 1. DIEU KIEN DUNG
        if (target == 0)
        {
            // Tim thay mot to hop co tong bang T. Ghi nhan ket qua.
            all_combinations.push_back(current_combination);
            return;
        }
        // Neu target < 0, nhanh nay khong the dan den loi giai hop le
        if (target < 0)
        {
            return;
```

```
        }
        // 2. VONG LAP (LUA CHON VA CAT TIA)
        // i la chi muc trong S
        for (int i = start_index; i < N; i++)
        {
            // Cat tia nhanh 1 (Pruning):
            // Neu S[i] lon hon target con lai, cac phan tu sau do
            // cung se lon hon (do S da sap xep), nen dung vong lap.
            if (S[i] > target)
            {
                break;
            }
            // Them S[i] vao to hop hien tai (Lua chon)
            current_combination.push_back(S[i]);
            // DE QUY (Chuyen trang thai)
            // target moi: target - S[i]
            // start_index moi: i + 1 (Dam bao S[i] khong duoc chon lai)
            solve(target - S[i], i + 1);
            // QUAY LUI (BACKTRACK)
            // Loai bo S[i] khoi to hop de thu lua chon tiep theo
            current_combination.pop_back();
        }
    };
    cout << "--- Sinh To hop co Tong bang " << T << " tu tap { ";
    for (int x : S) cout << x << " ";
    cout << "} ---" << endl;
    // Bat dau voi tong T, chi muc 0, to hop rong
    solve(T, 0);
    return all_combinations;
}
int main()
{
    // Vi du 1
    vector<int> S1 = {10, 1, 2, 7, 6, 1, 5};
    int T1 = 8;
    auto result1 = combination_sum_unique(S1, T1);
    cout << "Ket qua tim duoc: ";
    for (const auto& comb : result1)
    {
        cout << "[ ";
        for (int x : comb) cout << x << " ";
        cout << "] ";
    }
    cout << endl;
    // Vi du 2
    vector<int> S2 = {2, 3, 5};
    int T2 = 8;
    auto result2 = combination_sum_unique(S2, T2);
    cout << "Ket qua tim duoc: ";
    for (const auto& comb : result2)
    {
        cout << "[ ";
        for (int x : comb) cout << x << " ";
        cout << "] ";
    }
    cout << endl;
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
--- Sinh To hop co Tong bang 8 tu tap { 1 1 2 5 6 7 10 } ---
Ket qua tim duoc: [ 1 1 6 ] [ 1 2 5 ] [ 1 7 ] [ 1 2 5 ] [ 1 7 ] [ 2 6 ]
--- Sinh To hop co Tong bang 8 tu tap { 2 3 5 } ---
Ket qua tim duoc: [ 3 5 ]
```

BÀI 29. SINH HOÁN VỊ

I. Mục Tiêu:

- **Kiến thức:** HS nắm vững sự khác biệt giữa Hoán vị và Tổ hợp, và hiểu cách sử dụng mảng đánh dấu trạng thái (used array) để sinh Hoán vị.
- **Kỹ năng:** HS cài đặt được thuật toán quay lui để liệt kê tất cả các hoán vị của N phần tử trong Python và C++.
- **Tư duy:** HS phân tích được vai trò của thao tác **Quay lui (Backtrack/Undo)** trong việc khôi phục trạng thái sau khi gọi đệ quy.

II. Phân Bổ Thời Gian (90 Phút):

Hoạt Động	Nội Dung	Thời gian (Phút)
Ôn tập & Giới thiệu	Khái niệm Hoán vị, So sánh với Tổ hợp, Cây tìm kiếm Hoán vị	15
Lý thuyết	Trạng thái, Công thức Truy hồi và Mảng đánh dấu	25
Thực hành	Bài 1: Cài đặt Sinh Hoán vị O(N . N!) (Python & C++)	40
Tổng kết	Vai trò của Backtrack, Bài tập về nhà	10
Tổng cộng		**90**

III. Lý Thuyết 1: Hoán vị và Cấu trúc Quay lui (15 phút)

1. Định nghĩa Hoán vị (Permutation):

- **Hoán vị** là sự sắp xếp thứ tự của các phần tử trong một tập hợp. Khác với Tổ hợp (nơi thứ tự không quan trọng), trong Hoán vị, sự thay đổi về thứ tự tạo ra một kết quả khác.
- **Ví dụ:** Tập {1, 2, 3}.
  + **Hoán vị:** {1, 2, 3}, {1, 3, 2}, {2, 1, 3}, {2, 3, 1}, {3, 1, 2}, {3, 2, 1}. (Tổng cộng 3! = 6 hoán vị).
  + **Độ phức tạp:** Số lượng hoán vị của N phần tử là N! (N giai thừa).

2. Cấu trúc Quay lui cho Hoán vị:

Để sinh hoán vị, ta xây dựng dãy kết quả từng vị trí một (từ vị trí 1 đến N).

- **Mảng A:** Mảng kích thước N để lưu trữ hoán vị hiện tại.
- **Mảng Đánh dấu (used):** Mảng Boolean (hoặc số nguyên) kích thước N. used[i] = True nếu phần tử i đã được chọn và đặt vào một vị trí nào đó trong hoán vị.

IV. Lý thuyết 2: Công thức Truy hồi và Mảng Đánh dấu (25 phút)

- Ta định nghĩa hàm quay lui **solve(k)**:

k: Vị trí hiện tại ta đang tìm phần tử để đặt vào A[k] (từ 1 đến N).

- **Công thức Truy hồi:**

+ **Điều kiện Dừng (Base Case):** Nếu đã điền xong tất cả các vị trí (k = N), in ra hoán vị A hiện tại.

- **Lựa chọn và Đệ quy:** Ta thử các giá trị i (tức là các phần tử) từ 1 đến N:

+ **Ràng buộc:** Nếu phần tử i **chưa được sử dụng** (used[i] = False), ta có thể chọn nó:

▪ Đặt A[k] = i.

▪ **Đánh dấu:** used[i] = True.

▪ Gọi đệ quy: **solve(k + 1)**.

▪ **QUAY LUI (Backtrack):** Sau khi lời gọi đệ quy **solve(k + 1)** hoàn thành (và đã tìm thấy tất cả các hoán vị bắt đầu bằng **A[k] = i**), ta phải **khôi phục trạng thái ban đầu** để vòng lặp **for** có thể thử giá trị **i+1**. Ta hủy đánh dấu: **used[i] = False**.

V. Thực Hành: Cài đặt Sinh Hoán vị (40 phút)

Bài toán: Cho N, liệt kê tất cả các hoán vị của {1, 2, …, N}.

1. Code Python:

```
def permutations_backtracking(N):
    # Sử dụng thuật toán Quay lui (Backtracking) để liệt kê tất cả
    # các hoán vị của tập hợp {1, 2, ..., N}.
    # Độ phức tạp thời gian: O(N * N!), nơi N! là số lượng hoán vị.
    # Tham số:
    # N (int): Số lượng phần tử (từ 1 đến N).
    # Mảng 'permutation': Lưu trữ hoán vị hiện tại. Kích thước N (chỉ mục 0-dựa).
    permutation = [0] * N
    # Mảng 'used': Đánh dấu xem phần tử có giá trị i đã được sử dụng chưa.
    # Kích thước N+1 vì các phần tử là 1, 2, ..., N (sử dụng chỉ mục 1-dựa).
    used = [False] * (N + 1)
    def solve(k):
        # Hàm quay lui để điền giá trị vào vị trí k của hoán vị.
        # k (int): Chỉ mục của vị trí đang điền (từ 0 đến N-1).
        # 1. ĐIỀU KIỆN DỪNG (BASE CASE)
        if k == N:
            # Đã điền xong N vị trí, in ra hoán vị hoàn chỉnh
            print(permutation)
            return
        # 2. VÒNG LẶP (LỰA CHỌN)
        # Thử chọn các phần tử có giá trị từ 1 đến N
        for i in range(1, N + 1):
            # Ràng buộc: Chỉ chọn phần tử i nếu nó chưa được sử dụng
            if not used[i]:
                # A. Đặt giá trị và Đánh dấu (Tiến lên)
                permutation[k] = i  # Đặt giá trị i vào vị trí k
                used[i] = True      # Đánh dấu i đã được sử dụng
                # B. ĐỆ QUY (Chuyển trạng thái)
                # Điền vào vị trí tiếp theo (k + 1)
                solve(k + 1)
                # C. QUAY LUI (Backtrack)
                # Khôi phục trạng thái: Xóa đánh dấu i để nó có thể được chọn
                # lại cho các nhánh/hoán vị khác.
                used[i] = False
```

```python
    print(f"--- Hoan vi cua {N} phan tu ---")
    # Bắt đầu điền vào vị trí đầu tiên (k=0)
    solve(0)
# ---------------- CHẠY THỬ VÀ MINH HỌA ----------------
N = 3
permutations_backtracking(N)
# Kết quả: [1, 2, 3], [1, 3, 2], [2, 1, 3], [2, 3, 1], [3, 1, 2], [3, 2, 1]
```

=> Kết quả sau khi chạy chương trình:

```
--- Hoan vi cua 3 phan tu ---
[1, 2, 3]
[1, 3, 2]
[2, 1, 3]
[2, 3, 1]
[3, 1, 2]
[3, 2, 1]
```

- Giải thích Code Python:

+ Sử dụng used: Ta dùng mảng used kích thước N+1 để chỉ mục 1-dựa (i) có thể truy cập trực tiếp.

+ **Thao tác Quay lui (used[i] = False):** Đây là bước quan trọng nhất, đảm bảo khi hàm đệ quy trả về, trạng thái của cây tìm kiếm được khôi phục.

2. Code C++:

```cpp
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
using namespace std;         // Tiền tố std
void permutations_backtracking(int N)
{
    // Su dung thuat toan Quay lui (Backtracking) de liet ke tat ca
    // cac hoan vi cua tap hop {1, 2, ..., N}.
    // Do phuc tap thoi gian: O(N * N!), noi N! la so luong hoan vi.
    // Tham so:
    // N (int): So luong phan tu (tu 1 den N).
    // Mang 'permutation': Luu tru hoan vi hien tai. Kich thuoc N (chi muc 0-dua).
    vector<int> permutation(N);
    // Mang 'used': Danh dau xem phan tu co gia tri i da duoc su dung chua.
    // Kich thuoc N+1 vi cac phan tu la 1, 2, ..., N (su dung chi muc 1-dua).
    vector<bool> used(N + 1, false);
    // Ham quay lui
    function<void(int)> solve = [&](int k)
    {
        // Ham quay lui de dien gia tri vao vi tri k cua hoan vi.
        // k (int): Chi muc cua vi tri dang dien (tu 0 den N-1).
        // 1. DIEU KIEN DUNG (BASE CASE)
        if (k == N)
        {
            // Da dien xong N vi tri, in ra hoan vi hoan chinh
            cout << "[ ";
            for (int x : permutation)
            {
                cout << x << " ";
            }
            cout << "]" << endl;
            return;
        }
        // 2. VONG LAP (LUA CHON)
        // Thu chon cac phan tu co gia tri tu 1 den N
```

```
        for (int i = 1; i <= N; i++)
        {
            // Rang buoc: Chi chon phan tu i neu no chua duoc su dung
            if (!used[i])
            {
                // A. Dat gia tri va Danh dau (Tien len)
                permutation[k] = i; // Dat gia tri i vao vi tri k
                used[i] = true;     // Danh dau i da duoc su dung
                // B. DE QUY (Chuyen trang thai)
                // Dien vao vi tri tiep theo (k + 1)
                solve(k + 1);
                // C. QUAY LUI (Backtrack)
                // Khoi phuc trang thai: Xoa danh dau i de no co the duoc chon
                // lai cho cac nhanh/hoan vi khac.
                used[i] = false;
            }
        }
    };
    cout << "--- Hoan vi cua " << N << " phan tu ---" << endl;
    // Bat dau dien vao vi tri dau tien (k = 0)
    solve(0);
}
int main()
{
    int N = 3;
    permutations_backtracking(N);
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
--- Hoan vi cua 3 phan tu ---
[ 1 2 3 ]
[ 1 3 2 ]
[ 2 1 3 ]
[ 2 3 1 ]
[ 3 1 2 ]
[ 3 2 1 ]
```

- Giải thích Code C++:

vector<bool>: C++ sử dụng **vector<bool>** làm mảng đánh dấu. Logic giống hệt Python, tập trung vào việc đặt giá trị **(permutation[k] = i)**, đánh dấu **(used[i] = true)**, đệ quy, và quan trọng nhất là khôi phục trạng thái **(used[i] = false)**.

VI. Tổng kết & Bài tập Về nhà (10 phút)

1. Tóm tắt Nguyên lý Quay lui cho Hoán vị:

- **Sử dụng Vị trí và Giá trị: k** là vị trí đang điền, **i** là giá trị đang được thử.
- **Mảng Đánh dấu (used):** Đây là công cụ cần thiết để đảm bảo mỗi phần tử chỉ được sử dụng một lần trong một hoán vị cụ thể.
- **Tầm quan trọng của Backtrack:** Thao tác khôi phục trạng thái **(used[i] = False)** là bắt buộc. Nếu bỏ qua bước này, ta sẽ chỉ tìm được hoán vị đầu tiên và thuật toán sẽ dừng lại do tất cả các phần tử đều bị đánh dấu là đã sử dụng.

2. Bài tập Về nhà:

2.1. Bài tập 1 (Hoán vị của Chuỗi): Mở rộng thuật toán để sinh tất cả các hoán vị của một chuỗi ký tự bất kỳ (ví dụ: "ABC"). (Lưu ý: Nếu chuỗi có các ký tự lặp lại, cần thêm logic để loại bỏ hoán vị trùng lặp).

2.1.1. Code Python:

```python
def permutations_of_string(S):
    # Sử dụng thuật toán Quay lui (Backtracking) để sinh tất cả các hoán vị
    # của một chuỗi ký tự S, loại bỏ các hoán vị trùng lặp.
    # Tham số:
    # S (str): Chuỗi ký tự đầu vào.
    # Trả về:
    # list of str: Danh sách các hoán vị duy nhất.
    # BƯỚC 1: Chuẩn bị
    # Chuyển chuỗi thành list và sắp xếp để dễ dàng xử lý trùng lặp
    char_list = sorted(list(S))
    N = len(char_list)
    permutation = [''] * N
    used = [False] * N
    all_permutations = []
    def solve(k):
        # k là chỉ mục đang điền (0-dựa)
        # ĐIỀU KIỆN DỪNG
        if k == N:
            # Đã điền xong N vị trí, thêm hoán vị hoàn chỉnh vào danh sách
            all_permutations.append("".join(permutation))
            return
        # VÒNG LẶP (LỰA CHỌN) - i là chỉ mục trong char_list
        for i in range(N):
            # 1. RÀNG BUỘC SỬ DỤNG: Đảm bảo phần tử chưa được dùng
            if used[i]:
                continue
            # 2. RÀNG BUỘC TRÙNG LẶP: Xử lý Hoán vị của các Ký tự giống nhau
            # Điều kiện:
            # a) i > 0 (không phải phần tử đầu tiên)
            # b) char_list[i] == char_list[i-1] (giống phần tử liền trước)
            # c) NOT used[i-1] (Phần tử liền trước chưa được sử dụng)
            #    Điều này có nghĩa là ta đã bỏ qua nhánh của phần tử liền trước
            #    ở bước lựa chọn trước đó, và nếu ta chọn phần tử i, nó sẽ sinh ra
            #    một hoán vị trùng lặp với nhánh đã bị bỏ qua.
            if i > 0 and char_list[i] == char_list[i-1] and not used[i-1]:
                continue
            # 3. Đặt giá trị và Đánh dấu (Tiến lên)
            permutation[k] = char_list[i]
            used[i] = True
            # 4. ĐỆ QUY
            solve(k + 1)
            # 5. QUAY LUI (Backtrack)
            # Khôi phục trạng thái
            used[i] = False
    print(f"--- Hoan vi cua chuoi '{S}' ---")
    solve(0)
    return all_permutations
# ----------------- CHẠY THỬ VÀ MINH HỌA -----------------
# Ví dụ 1: Chuỗi không có ký tự lặp lại
S1 = "ABC"
# Kết quả mong đợi (3! = 6 hoán vị): ABC, ACB, BAC, BCA, CAB, CBA
result1 = permutations_of_string(S1)
print(f"Chuoi: '{S1}' -> {result1} (Tong: {len(result1)})")
# Ví dụ 2: Chuỗi có ký tự lặp lại
S2 = "AAB"
# Nếu không xử lý trùng lặp, ta sẽ sinh ra 6 hoán vị, nhưng chỉ 3 là duy nhất:
```

```
# A1 A2 B, A2 A1 B, A1 B A2, A2 B A1, B A1 A2, B A2 A1
# Kết quả duy nhất: AAB, ABA, BAA (3 hoán vị)
result2 = permutations_of_string(S2)
print(f"Chuoi: '{S2}' -> {result2} (Tong: {len(result2)})")
```

=> Kết quả sau khi chạy chương trình:

```
--- Hoan vi cua chuoi 'ABC' ---
Chuoi: 'ABC' -> ['ABC', 'ACB', 'BAC', 'BCA', 'CAB', 'CBA'] (Tong: 6)
--- Hoan vi cua chuoi 'AAB' ---
Chuoi: 'AAB' -> ['AAB', 'ABA', 'BAA'] (Tong: 3)
```

2.1.2. Code C++:

```
#include <iostream>          // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>     // Thư viện tổng hợp
using namespace std;         // Tiền tố std
vector<string> permutations_of_string(string S)
{
    // Su dung thuat toan Quay lui (Backtracking) de sinh tat ca cac hoan vi
    // cua mot chuoi ky tu S, loai bo cac hoan vi trung lap.
    // Tham so:
    // S (string): Chuoi ky tu dau vao.
    // Tra ve:
    // vector<string>: Danh sach cac hoan vi duy nhat.
    // BUOC 1: Chuan bi
    // Chuyen chuoi thanh vector ky tu va sap xep de de dang xu ly trung lap
    vector<char> char_list(S.begin(), S.end());
    sort(char_list.begin(), char_list.end());
    int N = char_list.size();
    vector<char> permutation(N);
    vector<bool> used(N, false);
    vector<string> all_permutations;
    function<void(int)> solve = [&](int k)
    {
        // k la chi muc dang dien (0-dua)
        // DIEU KIEN DUNG
        if (k == N)
        {
            // Da dien xong N vi tri, them hoan vi hoan chinh vao danh sach
            string temp(permutation.begin(), permutation.end());
            all_permutations.push_back(temp);
            return;
        }
        // VONG LAP (LUA CHON) - i la chi muc trong char_list
        for (int i = 0; i < N; i++)
        {
            // 1. RANG BUOC SU DUNG: Dam bao phan tu chua duoc dung
            if (used[i]) continue;
            // 2. RANG BUOC TRUNG LAP: Xu ly Hoan vi cua cac Ky tu giong nhau
            // Dieu kien:
            // a) i > 0 (khong phai phan tu dau tien)
            // b) char_list[i] == char_list[i-1] (giong phan tu lien truoc)
            // c) NOT used[i-1] (Phan tu lien truoc chua duoc su dung)
            if (i > 0 && char_list[i] == char_list[i - 1] && !used[i - 1])
                continue;
            // 3. Dat gia tri va Danh dau (Tien len)
            permutation[k] = char_list[i];
            used[i] = true;
            // 4. DE QUY
            solve(k + 1);
```

```
            // 5. QUAY LUI (Backtrack)
            // Khoi phuc trang thai
            used[i] = false;
        }
    };
    cout << "--- Hoan vi cua chuoi '" << S << "' ---" << endl;
    solve(0);
    return all_permutations;
}
int main()
{
    // Vi du 1: Chuoi khong co ky tu lap
    string S1 = "ABC";
    vector<string> result1 = permutations_of_string(S1);
    cout << "Chuoi: '" << S1 << "' -> ";
    for (auto &s : result1) cout << s << " ";
    cout << "(Tong: " << result1.size() << ")" << endl;
    // Vi du 2: Chuoi co ky tu lap
    string S2 = "AAB";
    vector<string> result2 = permutations_of_string(S2);
    cout << "Chuoi: '" << S2 << "' -> ";
    for (auto &s : result2) cout << s << " ";
    cout << "(Tong: " << result2.size() << ")" << endl;
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
--- Hoan vi cua chuoi 'ABC' ---
Chuoi: 'ABC' -> ['ABC', 'ACB', 'BAC', 'BCA', 'CAB', 'CBA'] (Tong: 6)
--- Hoan vi cua chuoi 'AAB' ---
Chuoi: 'AAB' -> ['AAB', 'ABA', 'BAA'] (Tong: 3)
```

2.2. Bài tập 2 (Hoán vị Sử dụng Hàm có sẵn): Nghiên cứu và sử dụng hàm thư viện có sẵn (ví dụ: **itertools.permutations** trong **Python**, **std::next_permutation** trong C++) để sinh hoán vị và so sánh kết quả với phương pháp Quay lui.

2.2.1. Code Python:

```
from itertools import permutations
import time
def generate_permutations_itertools(S):
    # Sinh tất cả các hoán vị của một tập hợp S sử dụng itertools.permutations.
    # Tham số:
    # S (list/str): Tập hợp hoặc chuỗi đầu vào.
    # Trả về:
    # list of tuples: Danh sách các hoán vị.
    # permutations(S) trả về một iterator của các hoán vị
    # list() được dùng để chuyển iterator thành danh sách
    return list(permutations(S))
# Lấy lại hàm Quay lui (Backtracking) để so sánh
def permutations_backtracking(S):
    # Sử dụng thuật toán Quay lui để sinh hoán vị (đã xử lý trùng lặp giá trị).
    char_list = sorted(list(S))
    N = len(char_list)
    permutation = [''] * N
    used = [False] * N
    all_permutations = []
    def solve(k):
        if k == N:
            all_permutations.append("".join(permutation))
```

```
            return
        for i in range(N):
            if used[i]:
                continue
            # Xử lý trùng lặp (chỉ áp dụng nếu S có ký tự lặp)
            if i > 0 and char_list[i] == char_list[i-1] and not used[i-1]:
                continue
            permutation[k] = char_list[i]
            used[i] = True
            solve(k + 1)
            used[i] = False
    solve(0)
    return all_permutations
# ----------------- CHẠY THỬ VÀ SO SÁNH -----------------
S_input = "CAT"
S_repeat = "AAAB"
print("==========================================")
print("1. SO SANH PHUONG PHAP (Khong co ky tu lap)")
print(f"Chuoi dau vao: '{S_input}'")
# A. Phương pháp Itertools
start_time_it = time.time()
result_it = generate_permutations_itertools(S_input)
end_time_it = time.time()
print(f"\n[A] Itertools.permutations (Thoi gian: {end_time_it -
start_time_it:.6f}s)")
# Kết quả của Itertools là tuple, ta chuyển sang list để dễ so sánh
print(list(map(lambda x: "".join(x), result_it)))
# B. Phương pháp Quay lui
start_time_bt = time.time()
result_bt = permutations_backtracking(S_input)
end_time_bt = time.time()
print(f"\n[B] Quay lui (Thoi gian: {end_time_bt - start_time_bt:.6f}s)")
print(result_bt)
print("\n==========================================")
print("2. SO SANH KHI CO KY TU LAP LAI (Xu ly Duy nhat)")
print(f"Chuoi dau vao: '{S_repeat}'")
# C. Phương pháp Itertools
# Itertools KHÔNG TỰ ĐỘNG xử lý trùng lặp.
# Ta phải chuyển kết quả sang set() để lọc trùng.
start_time_it_rep = time.time()
result_it_rep = generate_permutations_itertools(S_repeat)
end_time_it_rep = time.time()
# Lọc trùng lặp bằng set
unique_it = list(set(map("".join, result_it_rep)))
print(f"\n[C] Itertools + Loc Set (Thoi gian: {end_time_it_rep -
start_time_it_rep:.6f}s)")
print(unique_it)
# D. Phương pháp Quay lui (có xử lý trùng lặp)
start_time_bt_rep = time.time()
result_bt_rep = permutations_backtracking(S_repeat)
end_time_bt_rep = time.time()
print(f"\n[D] Quay lui (co logic xu ly trung lap) (Thoi gian: {end_time_bt_rep -
start_time_bt_rep:.6f}s)")
print(result_bt_rep)
```

=> Kết quả sau khi chạy chương trình:

```
1. SO SANH PHUONG PHAP (Khong co ky tu lap)
Chuoi dau vao: 'CAT'

[A] Itertools.permutations (Thoi gian: 0.000008s)
['CAT', 'CTA', 'ACT', 'ATC', 'TCA', 'TAC']

[B] Quay lui (Thoi gian: 0.000410s)
['ACT', 'ATC', 'CAT', 'CTA', 'TAC', 'TCA']

==========================================
2. SO SANH KHI CO KY TU LAP LAI (Xu ly Duy nhat)
Chuoi dau vao: 'AAAB'

[C] Itertools + Loc Set (Thoi gian: 0.000132s)
['AABA', 'ABAA', 'BAAA', 'AAAB']

[D] Quay lui (co logic xu ly trung lap) (Thoi gian: 0.000036s)
['AAAB', 'AABA', 'ABAA', 'BAAA']
```

2.2.2. Code C++:

```cpp
#include <iostream>             // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>        // Thư viện tổng hợp
#include <chrono>               // Thư viện đo thời gianchạy của Code
using namespace std;            // Tiền tố std
using namespace std::chrono;    // Đo thời gian chạy Code
// ---------------------------------------------------
// A. Sinh hoan vi bang "thu vien" (tuong duong itertools)
// ---------------------------------------------------
vector<string> generate_permutations_stl(string S)
{
    // Sinh tat ca cac hoan vi cua mot tap hop S su dung STL.
    // Tham so:
    // S (string): Chuoi dau vao.
    // Tra ve:
    // vector<string>: Danh sach cac hoan vi.
    vector<string> result;
    // Sap xep truoc de next_permutation sinh day du
    sort(S.begin(), S.end());
    do
    {
        result.push_back(S);
    } while (next_permutation(S.begin(), S.end()));
    return result;
}
// ---------------------------------------------------
// B. Sinh hoan vi bang Quay lui (co xu ly trung lap)
// ---------------------------------------------------
vector<string> permutations_backtracking(string S)
{
    // Su dung thuat toan Quay lui de sinh hoan vi
    // (da xu ly trung lap gia tri).
    vector<char> char_list(S.begin(), S.end());
    sort(char_list.begin(), char_list.end());
    int N = char_list.size();
    vector<char> permutation(N);
    vector<bool> used(N, false);
    vector<string> all_permutations;
```

```
    function<void(int)> solve = [&](int k)
    {
        if (k == N)
        {
            all_permutations.push_back(
                string(permutation.begin(), permutation.end))
            );
            return;
        }
        for (int i = 0; i < N; i++)
        {
            if (used[i]) continue;
            // Xu ly trung lap (chi ap dung neu S co ky tu lap)
            if (i > 0 && char_list[i] == char_list[i - 1] && !used[i - 1])
                continue;
            permutation[k] = char_list[i];
            used[i] = true;
            solve(k + 1);
            used[i] = false;
        }
    };
    solve(0);
    return all_permutations;
}
// ---------------------------------------------------
// CHAY THU VA SO SANH
// ---------------------------------------------------
int main()
{
    string S_input = "CAT";
    string S_repeat = "AAAB";
    cout << "=========================================" << endl;
    cout << "1. SO SANH PHUONG PHAP (Khong co ky tu lap)" << endl;
    cout << "Chuoi dau vao: '" << S_input << "'" << endl;
    // A. Phuong phap STL (tuong duong itertools)
    auto start_it = high_resolution_clock::now();
    vector<string> result_it = generate_permutations_stl(S_input);
    auto end_it = high_resolution_clock::now();
    cout << "\n[A] STL next_permutation (Thoi gian: "
         << duration<double>(end_it - start_it).count() << "s)" << endl;
    for (auto &s : result_it) cout << s << " ";
    cout << endl;
    // B. Phuong phap Quay lui
    auto start_bt = high_resolution_clock::now();
    vector<string> result_bt = permutations_backtracking(S_input);
    auto end_bt = high_resolution_clock::now();
    cout << "\n[B] Quay lui (Thoi gian: "
         << duration<double>(end_bt - start_bt).count() << "s)" << endl;
    for (auto &s : result_bt) cout << s << " ";
    cout << endl;
    cout << "\n=========================================" << endl;
    cout << "2. SO SANH KHI CO KY TU LAP LAI (Xu ly Duy nhat)" << endl;
    cout << "Chuoi dau vao: '" << S_repeat << "'" << endl;
    // C. STL + loc trung bang set
    auto start_it_rep = high_resolution_clock::now();
    vector<string> result_it_rep = generate_permutations_stl(S_repeat);
    auto end_it_rep = high_resolution_clock::now();
    set<string> unique_it(result_it_rep.begin(), result_it_rep.end());
```

```
    cout << "\n[C] STL + Loc set (Thoi gian: "
         << duration<double>(end_it_rep - start_it_rep).count() << "s)" << endl;
    for (auto &s : unique_it) cout << s << " ";
    cout << endl;
    // D. Quay lui co xu ly trung lap
    auto start_bt_rep = high_resolution_clock::now();
    vector<string> result_bt_rep = permutations_backtracking(S_repeat);
    auto end_bt_rep = high_resolution_clock::now();
    cout << "\n[D] Quay lui (co logic xu ly trung lap) (Thoi gian: "
         << duration<double>(end_bt_rep - start_bt_rep).count() << "s)" << endl;
    for (auto &s : result_bt_rep) cout << s << " ";
    cout << endl;
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
1. SO SANH PHUONG PHAP (Khong co ky tu lap)
Chuoi dau vao: 'CAT'

[A] STL next_permutation (Thoi gian: 5.29e-05s)
ACT ATC CAT CTA TAC TCA

[B] Quay lui (Thoi gian: 3.75e-05s)
ACT ATC CAT CTA TAC TCA

==========================================
2. SO SANH KHI CO KY TU LAP LAI (Xu ly Duy nhat)
Chuoi dau vao: 'AAAB'

[C] STL + Loc set (Thoi gian: 6.8e-06s)
AAAB AABA ABAA BAAA

[D] Quay lui (co logic xu ly trung lap) (Thoi gian: 1.45e-05s)
AAAB AABA ABAA BAAA
```

BÀI 30. BÀI TOÁN PHÂN CHIA

I. Mục Tiêu:

- **Kiến thức:** HS hiểu cách áp dụng kỹ thuật Quay lui để giải quyết các bài toán phân chia tập hợp (Partitioning), một dạng bài toán tìm kiếm có ràng buộc.
- **Kỹ năng:** HS cài đặt được thuật toán quay lui để kiểm tra xem một tập hợp có thể được chia thành hai tập con có tổng bằng nhau hay không.
- **Tư duy:** HS phân tích được cây tìm kiếm (Search Tree) trong bài toán phân chia, xác định được trạng thái đệ quy cần thiết (chỉ mục, tổng hiện tại, v.v.).

II. Phân Bổ Thời Gian (90 Phút):

Hoạt Động	Nội Dung	Thời gian (Phút)
Ôn tập & Giới thiệu	Định nghĩa Bài toán Phân chia Tập hợp, điều kiện cần	15

Lý thuyết	Trạng thái, Công thức Truy hồi và Cắt tỉa (Pruning)	25
Thực hành	Bài 1. Kiểm tra Phân chia Tập con Tổng bằng nhau (Python & C++)	40
Tổng kết	So sánh với DP, Bài tập về nhà	10
Tổng cộng		**90**

III. Lý Thuyết 1: Giới thiệu Bài toán Phân chia (15 phút)

1. Bài toán Phân chia Tập hợp (Set Partition Problem):

Cho một tập hợp **S** các số nguyên, bài toán phân chia tập hợp yêu cầu chia **S** thành **m** tập con rời rạc (S_1, S_2, …, S_m) sao cho tổng của các phần tử trong mỗi tập con thỏa mãn một điều kiện nhất định.

2. Bài toán trọng tâm: Chia thành hai tập con có tổng bằng nhau

- **Mục tiêu:** Kiểm tra xem tập **S** có thể được chia thành hai tập con $\mathbf{S_1}$ và $\mathbf{S_2}$ sao cho $\sum_{x \in S_1} x = \sum_{y \in S_2} y$ hay không.
- **Điều kiện cần (Necessity):**
  + Tổng tất cả các phần tử trong **S (sum(S))** phải là số chẵn.
  + Tổng mục tiêu (Target Sum) cho mỗi tập con là: **T = sum(S) / 2**.
- **Quy về bài toán con:** Bài toán trở thành: **"Tìm một tập con $\mathbf{S_1}$ của S sao cho tổng các phần tử của nó bằng T."**

3. Áp dụng Quay lui:

Ta sẽ duyệt qua từng phần tử của **S** và tại mỗi phần tử, ta có hai lựa chọn (giống cây quyết định nhị phân):

- **Lựa chọn 1 (Bao gồm):** Thêm phần tử hiện tại vào tập con $\mathbf{S_1}$ và giảm tổng mục tiêu **T**.
- **Lựa chọn 2 (Loại trừ):** Bỏ qua phần tử hiện tại và giữ nguyên tổng mục tiêu **T**.

IV. Lý thuyết 2: Công thức Truy hồi và Cắt tỉa (25 phút)

- Ta định nghĩa hàm quay lui **solve(index, target)**:
  + index: Chỉ mục của phần tử hiện tại đang được xét trong mảng **S**.
  + target: Tổng còn lại cần đạt được cho tập con $\mathbf{S_1}$.
- **Công thức Truy hồi:**
  + Điều kiện Dừng (Base Case):
    ▪ Nếu **target = 0**: **Thành công** (Đã tìm thấy tập con $\mathbf{S_1}$ có tổng bằng **T**). Trả về **True**.
    ▪ Nếu **target < 0** hoặc **index = N**: **Thất bại** (Không thể đạt được tổng **T** từ nhánh này). Trả về **False**.
  + Lựa chọn và Đệ quy:
    ▪ Lựa chọn 1 (Bao gồm **S[index]**):
      Bao gồm = solve(index + 1, target - S[index])
    ▪ Lựa chọn 2 (Loại trừ **S[index]**):
      Loại trừ = solve(index + 1, target)
    ▪ **Kết quả:** Trả về **Bao gồm OR Loại trừ**.
  + Cắt tỉa (Pruning):

- Sắp xếp tập **S** (tăng dần).
- Khi đệ quy, nếu **S[index] > target**, ta không cần thử lựa chọn **Bao gồm** (vì nó chắc chắn làm cho **target** trở thành số âm). Ta có thể dừng nhánh đó.

V. Thực Hành: Kiểm tra Phân chia Tập con Tổng bằng nhau (40 phút)

1. Code Python:

```
def can_partition(S):
    # Kiểm tra xem tập hợp S có thể được chia thành hai tập con có tổng bằng nhau
hay không.
    # Bài toán được quy về tìm kiếm một tập con có tổng bằng Target = Tổng(S) /
2.
    # Tham số:
    # S (list): Tập hợp các số nguyên đầu vào.
    # Trả về:
    # bool: True nếu có thể phân chia, False nếu ngược lại.
    total_sum = sum(S)
    # Điều kiện cần: Tổng tất cả các phần tử phải là số chẵn.
    if total_sum % 2 != 0:
        return False
    # Tổng mục tiêu (Target Sum) cho mỗi tập con
    target = total_sum // 2
    N = len(S)
    # Tối ưu: Sắp xếp giảm dần.
    # Việc này giúp việc cắt tỉa (khi current_target < 0) diễn ra sớm hơn,
    # vì ta sẽ thử các phần tử lớn trước.
    S.sort(reverse=True)
    def solve(index, current_target):
        # Hàm quay lui (DFS) tìm kiếm tập con có tổng bằng current_target.
        # Tham số:
        # index (int): Chỉ mục của phần tử S[index] đang được xét.
        # current_target (int): Tổng còn lại cần đạt được.
        # 1. ĐIỀU KIỆN DỪNG THÀNH CÔNG (BASE CASE)
        if current_target == 0:
            # Tìm thấy một tập con có tổng bằng Target
            return True
        # 2. ĐIỀU KIỆN DỪNG THẤT BẠI
        # Hết phần tử để xét HOẶC Tổng còn lại đã bị âm (vượt quá mục tiêu)
        if index == N or current_target < 0:
            return False
        # Cắt tỉa (Pruning): Nếu S[index] lớn hơn tổng mục tiêu còn lại
        # Không cần thử nhánh Bao gồm, vì S đã sắp xếp giảm dần.
        if S[index] > current_target:
            # Chuyển sang lựa chọn loại trừ ngay lập tức
            # return solve(index + 1, current_target) # Có thể viết tường minh hoặc
dựa vào logic bên dưới
            pass # Sẽ được xử lý bởi Lựa chọn 2
        # 3. CÂY QUYẾT ĐỊNH (LỰA CHỌN)
        # Lựa chọn 1: Bao gồm S[index] vào tập con S1 (Giảm target)
        # NẾU nhánh này tìm thấy lời giải, trả về True ngay lập tức
        if solve(index + 1, current_target - S[index]):
            return True
        # Lựa chọn 2: Loại trừ S[index] khỏi tập con S1 (Giữ target)
        # Nếu Lựa chọn 1 thất bại, ta thử Lựa chọn 2
        return solve(index + 1, current_target)
    # Bắt đầu quay lui từ index 0 với target ban đầu
    return solve(0, target)
```

```
# ----------------- CHẠY THỬ VÀ MINH HỌA -----------------
S1 = [1, 5, 11, 5]
# Tổng = 22, Target = 11. Tập con 1: [11], Tập con 2: [1, 5, 5].
print(f"Tap hop S1: {S1}")
print(f"Can Partition: {can_partition(S1)}") # Kết quả: True
S2 = [1, 5, 3]
# Tổng = 9, Target = 4.5. Không thể chia.
print(f"\nTap hop S2: {S2}")
print(f"Can Partition: {can_partition(S2)}") # Kết quả: False
```

=> Kết quả sau khi chạy chương trình:

```
Tap hop S1: [1, 5, 11, 5]
Can Partition: True

Tap hop S2: [1, 5, 3]
Can Partition: False
```

- Giải thích Code:

+ Sắp xếp giảm dần: Giúp việc cắt tỉa **(current_target < 0)** diễn ra sớm hơn, tối ưu tốc độ tìm kiếm.

+ Hàm đệ quy: Trả về True ngay khi tìm thấy lời giải đầu tiên, vì ta chỉ cần kiểm tra khả năng tồn tại chứ không cần liệt kê.

2. Code C++:

```
#include <iostream>             // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>        // Thư viện tổng hợp
using namespace std;            // Tiền tố std
bool can_partition(vector<int> S)
{
    // Kiem tra xem tap hop S co the duoc chia thanh hai tap con
    // co tong bang nhau hay khong.
    // Bai toan duoc quy ve tim kiem mot tap con
    // co tong bang Target = Tong(S) / 2.
    //
    // Tham so:
    // S (vector<int>): Tap hop cac so nguyen dau vao.
    //
    // Tra ve:
    // bool: True neu co the phan chia, False neu nguoc lai.
    int total_sum = 0;
    for (int x : S) total_sum += x;
    // Dieu kien can: Tong tat ca cac phan tu phai la so chan.
    if (total_sum % 2 != 0)
        return false;
    // Tong muc tieu (Target Sum) cho moi tap con
    int target = total_sum / 2;
    int N = S.size();
    // Toi uu: Sap xep giam dan.
    // Viec nay giup viec cat tia (khi current_target < 0)
    // dien ra som hon, vi ta se thu cac phan tu lon truoc.
    sort(S.begin(), S.end(), greater<int>());
    // Ham quay lui (DFS) tim kiem tap con co tong bang current_target
    function<bool(int, int)> solve = [&](int index, int current_target) -> bool
    {
        // Tham so:
        // index (int): Chi muc cua phan tu S[index] dang duoc xet
        // current_target (int): Tong con lai can dat duoc
        // 1. DIEU KIEN DUNG THANH CONG (BASE CASE)
```

```
        if (current_target == 0)
        {
            // Tim thay mot tap con co tong bang Target
            return true;
        }
        // 2. DIEU KIEN DUNG THAT BAI
        // Het phan tu de xet HOAC Tong con lai da bi am
        if (index == N || current_target < 0)
        {
            return false;
        }
        // Cat tia (Pruning): Neu S[index] lon hon tong muc tieu con lai
        // Khong can thu nhanh Bao gom, vi S da sap xep giam dan
        if (S[index] > current_target)
        {
            // Chuyen sang lua chon loai tru ngay lap tuc
            // return solve(index + 1, current_target);
            // Se duoc xu ly boi Lua chon 2
        }
        // 3. CAY QUYET DINH (LUA CHON)
        // Lua chon 1: Bao gom S[index] vao tap con S1
        // Neu nhanh nay tim thay loi giai, tra ve True ngay
        if (solve(index + 1, current_target - S[index]))
        {
            return true;
        }
        // Lua chon 2: Loai tru S[index] khoi tap con S1
        return solve(index + 1, current_target);
    };
    // Bat dau quay lui tu index 0 voi target ban dau
    return solve(0, target);
}
// ----------------- CHAY THU VA MINH HOA -----------------
int main()
{
    vector<int> S1 = {1, 5, 11, 5};
    // Tong = 22, Target = 11
    // Tap con 1: [11], Tap con 2: [1, 5, 5]
    cout << "Tap hop S1: ";
    for (int x : S1) cout << x << " ";
    cout << endl;
    cout << "Can Partition: " << can_partition(S1) << endl; // True
    vector<int> S2 = {1, 5, 3};
    // Tong = 9, Target = 4.5
    // Khong the chia
    cout << "\nTap hop S2: ";
    for (int x : S2) cout << x << " ";
    cout << endl;
    cout << "Can Partition: " << can_partition(S2) << endl; // False
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
Tap hop S1: 1 5 11 5
Can Partition: 1

Tap hop S2: 1 5 3
Can Partition: 0
```

VI. Tổng kết & Bài tập Về nhà (10 phút):

1. Tóm tắt Nguyên lý Quay lui cho Phân chia:

- **Chuyển đổi bài toán:** Phân chia thành hai tập con tổng bằng nhau => Tìm một tập con có tổng **T = sum(S) / 2**.
- **Cấu trúc Nhị phân:** Tại mỗi phần tử, có 2 lựa chọn: **Bao gồm** (giảm **target**) hoặc **Loại trừ** (giữ **target**).
- **Cắt tỉa (Pruning):** Sắp xếp và kiểm tra điều kiện **S[index] > target** giúp loại bỏ các nhánh vô ích.

2. Bài tập Về nhà:

2.1. Bài tập 1 (Phân chia K tập con): Mở rộng thuật toán để kiểm tra xem một tập hợp **S** có thể được chia thành **K tập con** rời rạc có tổng bằng nhau hay không. (Gợi ý: Cần một mảng kích thước **K** để theo dõi tổng hiện tại của mỗi nhóm).

2.1.1. Code Python:

```python
def can_partition_k_subsets(S, K):
    # Kiểm tra xem tập hợp S có thể được chia thành K tập con rời rạc có tổng bằng
nhau hay không.
    # Tham số:
    # S (list): Tập hợp các số nguyên đầu vào.
    # K (int): Số lượng tập con cần chia.
    # Trả về:
    # bool: True nếu có thể phân chia, False nếu ngược lại.
    N = len(S)
    if K <= 0 or N < K:
        return False
    total_sum = sum(S)
    # 1. KIỂM TRA ĐIỀU KIỆN CẦN VÀ TARGET
    if total_sum % K != 0:
        return False
    target = total_sum // K
    # Nếu bất kỳ phần tử nào lớn hơn target, không thể chia được
    if any(x > target for x in S):
        return False
    # Tối ưu: Sắp xếp giảm dần
    S.sort(reverse=True)
    # Khởi tạo mảng K nhóm (mỗi nhóm ban đầu có tổng bằng 0)
    subsets_sum = [0] * K
    def solve(index):
        # Hàm quay lui để quyết định phần tử S[index] thuộc nhóm nào.
        # index (int): Chỉ mục của phần tử S đang được xét.
        # 1. ĐIỀU KIỆN DỪNG THÀNH CÔNG
        # Nếu đã xét xong tất cả các phần tử (index = N)
        if index == N:
            # Nếu tất cả các nhóm đều đạt Target, thì thành công
            # (Thực ra chỉ cần kiểm tra nhóm cuối cùng bằng Target là đủ,
            # do tổng các nhóm = tổng S)
            return all(s == target for s in subsets_sum)
        # 2. VÒNG LẶP (LỰA CHỌN)
        # Thử thêm S[index] vào từng nhóm (từ 0 đến K-1)
        for j in range(K):
            # Ràng buộc/Cắt tỉa 1: Chỉ thêm vào nhóm nếu tổng mới <= Target
            if subsets_sum[j] + S[index] <= target:
                # A. Đặt giá trị (Tiến lên)
                subsets_sum[j] += S[index]
                # B. ĐỆ QUY (Chuyển trạng thái)
```

```
                # Chuyển sang xét phần tử tiếp theo (index + 1)
                if solve(index + 1):
                    return True # Tìm thấy lời giải
                # C. QUAY LUI (Backtrack)
                # Khôi phục trạng thái: loại bỏ S[index] khỏi nhóm j
                subsets_sum[j] -= S[index]
            # Cắt tỉa 2: Nếu nhóm hiện tại (j) chưa có phần tử nào
            # và không thể chứa S[index] (vì S[index] > target),
            # hoặc nếu nhóm j đã đầy (subsets_sum[j] == 0),
            # ta có thể dừng thử các nhóm tiếp theo.
            # (Đây là kỹ thuật nâng cao, đảm bảo không thử các hoán vị nhóm trùng
lặp)
            if subsets_sum[j] == 0:
                break
        return False
    print(f"--- Kiem tra Phan chia {K} nhom cho {S} (Target: {target}) ---")
    return solve(0)
# ---------------- CHẠY THỬ VÀ MINH HỌA ----------------
# Ví dụ 1: Chia 4 nhóm (K=4)
S1 = [4, 3, 2, 3, 5, 2, 1]
K1 = 4
# Tổng = 20. Target = 20/4 = 5.
# Kết quả mong đợi: [4, 1], [5], [3, 2], [3, 2]. -> True
result1 = can_partition_k_subsets(S1, K1)
print(f"S1: {S1}, K={K1} -> Can Partition: {result1}")
# Ví dụ 2: Không thể chia
S2 = [1, 2, 3, 4, 5, 6]
K2 = 3
# Tổng = 21. Target = 21/3 = 7.
# Có thể: [1, 6], [2, 5], [3, 4]. -> True
result2 = can_partition_k_subsets(S2, K2)
print(f"\nS2: {S2}, K={K2} -> Can Partition: {result2}")
# Ví dụ 3: Không thể chia
S3 = [1, 1, 1, 1, 2, 2, 2, 2]
K3 = 4
# Tổng = 12. Target = 12/4 = 3.
# Hoán vị là [2, 1], [2, 1], [2, 1], [2, 1]. -> True
result3 = can_partition_k_subsets(S3, K3)
print(f"\nS3: {S3}, K={K3} -> Can Partition: {result3}")
# Ví dụ 4: Thất bại do tổng lẻ
S4 = [1, 2, 3, 4]
K4 = 3
# Tổng = 10. Không chia hết cho 3.
result4 = can_partition_k_subsets(S4, K4)
print(f"\nS4: {S4}, K={K4} -> Can Partition: {result4}")
```

=> Kết quả sau khi chạy chương trình:

```
--- Kiem tra Phan chia 4 nhom cho [5, 4, 3, 3, 2, 2, 1] (Target: 5) ---
S1: [5, 4, 3, 3, 2, 2, 1], K=4 -> Can Partition: True
--- Kiem tra Phan chia 3 nhom cho [6, 5, 4, 3, 2, 1] (Target: 7) ---

S2: [6, 5, 4, 3, 2, 1], K=3 -> Can Partition: True
--- Kiem tra Phan chia 4 nhom cho [2, 2, 2, 2, 1, 1, 1, 1] (Target: 3) ---

S3: [2, 2, 2, 2, 1, 1, 1, 1], K=4 -> Can Partition: True

S4: [1, 2, 3, 4], K=3 -> Can Partition: False
```

2.1.2. Code C++:

```
#include <iostream>             // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>        // Thư viện tổng hợp
using namespace std;            // Tiền tố std
bool can_partition_k_subsets(vector<int> S, int K)
{
    // Kiem tra xem tap hop S co the duoc chia thanh K tap con roi rac
    // co tong bang nhau hay khong.
    //
    // Tham so:
    // S (vector<int>): Tap hop cac so nguyen dau vao.
    // K (int): So luong tap con can chia.
    //
    // Tra ve:
    // bool: True neu co the phan chia, False neu nguoc lai.
    int N = S.size();
    if (K <= 0 || N < K)
    {
        return false;
    }
    int total_sum = 0;
    for (int x : S) total_sum += x;
    // 1. KIEM TRA DIEU KIEN CAN VA TARGET
    if (total_sum % K != 0)
    {
        return false;
    }
    int target = total_sum / K;
    // Neu bat ky phan tu nao lon hon target, khong the chia duoc
    for (int x : S)
    {
        if (x > target)
        {
            return false;
        }
    }
    // Toi uu: Sap xep giam dan
    sort(S.begin(), S.end(), greater<int>());
    // Khoi tao mang K nhom (moi nhom ban dau co tong bang 0)
    vector<int> subsets_sum(K, 0);
    // Ham quay lui de quyet dinh phan tu S[index] thuoc nhom nao
    function<bool(int)> solve = [&](int index) -> bool
    {
        // index (int): Chi muc cua phan tu S dang duoc xet
        // 1. DIEU KIEN DUNG THANH CONG
        // Neu da xet xong tat ca cac phan tu
        if (index == N)
        {
            // Neu tat ca cac nhom deu dat Target thi thanh cong
            for (int s : subsets_sum)
            {
                if (s != target) return false;
            }
            return true;
        }
        // 2. VONG LAP (LUA CHON)
        // Thu them S[index] vao tung nhom (tu 0 den K-1)
```

```
        for (int j = 0; j < K; j++)
        {
            // Rang buoc / Cat tia 1:
            // Chi them vao nhom neu tong moi <= Target
            if (subsets_sum[j] + S[index] <= target)
            {
                // A. Dat gia tri (Tien len)
                subsets_sum[j] += S[index];
                // B. DE QUY (Chuyen trang thai)
                if (solve(index + 1))
                {
                    return true; // Tim thay loi giai
                }
                // C. QUAY LUI (Backtrack)
                subsets_sum[j] -= S[index];
            }
            // Cat tia 2:
            // Neu nhom hien tai chua co phan tu nao
            // thi khong can thu cac nhom tiep theo
            if (subsets_sum[j] == 0)
            {
                break;
            }
        }
        return false;
    };
    cout << "--- Kiem tra Phan chia " << K
         << " nhom cho tap S (Target: " << target << ") ---" << endl;
    return solve(0);
}
// ---------------- CHAY THU VA MINH HOA ----------------
int main()
{
    // Vi du 1
    vector<int> S1 = {4, 3, 2, 3, 5, 2, 1};
    int K1 = 4;
    bool result1 = can_partition_k_subsets(S1, K1);
    cout << "S1: ";
    for (int x : S1) cout << x << " ";
    cout << ", K=" << K1 << " -> Can Partition: " << result1 << endl;
    // Vi du 2
    vector<int> S2 = {1, 2, 3, 4, 5, 6};
    int K2 = 3;
    bool result2 = can_partition_k_subsets(S2, K2);
    cout << "\nS2: ";
    for (int x : S2) cout << x << " ";
    cout << ", K=" << K2 << " -> Can Partition: " << result2 << endl;
    // Vi du 3
    vector<int> S3 = {1, 1, 1, 1, 2, 2, 2, 2};
    int K3 = 4;
    bool result3 = can_partition_k_subsets(S3, K3);
    cout << "\nS3: ";
    for (int x : S3) cout << x << " ";
    cout << ", K=" << K3 << " -> Can Partition: " << result3 << endl;
    // Vi du 4
    vector<int> S4 = {1, 2, 3, 4};
    int K4 = 3;
    bool result4 = can_partition_k_subsets(S4, K4);
```

```
    cout << "\nS4: ";
    for (int x : S4) cout << x << " ";
    cout << ", K=" << K4 << " -> Can Partition: " << result4 << endl;
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
--- Kiem tra Phan chia 4 nhom cho tap S (Target: 5) ---
S1: 4 3 2 3 5 2 1 , K=4 -> Can Partition: 1
--- Kiem tra Phan chia 3 nhom cho tap S (Target: 7) ---

S2: 1 2 3 4 5 6 , K=3 -> Can Partition: 1
--- Kiem tra Phan chia 4 nhom cho tap S (Target: 3) ---

S3: 1 1 1 1 2 2 2 2 , K=4 -> Can Partition: 1

S4: 1 2 3 4 , K=3 -> Can Partition: 0
```

2.2. Bài tập 2 (So sánh với DP): Nghiên cứu thuật toán Quy hoạch động (DP) để giải quyết bài toán Subset Sum (tương đương với bài toán này) và so sánh hiệu suất giữa DP và Backtracking.

2.2.1. Code Python:

```
def subset_sum_dp(S, T):
    # Sử dụng Quy hoạch động để kiểm tra Subset Sum (tối ưu hóa không gian).
    # Độ phức tạp: O(N * T).
    N = len(S)
    # DP[j] = True nếu có thể tạo ra tổng j
    dp = [False] * (T + 1)
    dp[0] = True  # Có thể tạo ra tổng 0 (bằng tập rỗng)
    for num in S:
        # Duyệt ngược để đảm bảo mỗi num chỉ được sử dụng một lần
        for j in range(T, num - 1, -1):
            if dp[j - num]:
                dp[j] = True
    return dp[T]
def subset_sum_backtracking(S, T):
    # Sử dụng thuật toán Quay lui để kiểm tra Subset Sum.
    # Độ phức tạp: O(2^N).
    N = len(S)
    # Sắp xếp để hỗ trợ cắt tỉa (pruning)
    S.sort(reverse=True)
    def solve(index, current_target):
        if current_target == 0:
            return True
        if index == N or current_target < 0:
            return False
        # Cắt tỉa: Nếu phần tử hiện tại quá lớn, bỏ qua các phần tử nhỏ hơn
        if S[index] > current_target:
            return solve(index + 1, current_target)
        # Lựa chọn 1: Bao gồm S[index]
        if solve(index + 1, current_target - S[index]):
            return True
        # Lựa chọn 2: Loại trừ S[index]
        return solve(index + 1, current_target)
    return solve(0, T)
import time
import random
def run_comparison(N, T):
    # Tạo tập dữ liệu: Các số ngẫu nhiên sao cho tổng có thể đạt được T
```

```
    S = [random.randint(1, T // N + 1) for _ in range(N)]
    S.append(T - sum(S)) # Chèn thêm phần tử để tổng luôn là T
    # Trường hợp T lớn (T >> 2^N)
    if N <= 25 and T > 1000:
        pass # Sử dụng S đã tạo
    # Trường hợp N lớn (2^N >> T)
    elif N > 25 and T <= 500:
        # Nếu N quá lớn, Quay lui sẽ thất bại nhanh.
        # Tạo S nhỏ hơn để Quay lui có cơ hội chạy hết
        S = [random.randint(1, T // N + 1) for _ in range(25)]
        N = 25
        S.append(T - sum(S))
    # Đảm bảo S không chứa số âm sau khi chỉnh sửa
    S = [x for x in S if x > 0]
    T_final = sum(S) // 2 # Target cho bài toán Can Partition
    print(f"\n--- So sanh voi N={len(S)} va Target={T_final} ---")
    # --- Quay lui (O(2^N)) ---
    start_bt = time.time()
    result_bt = subset_sum_backtracking(S, T_final)
    end_bt = time.time()
    time_bt = end_bt - start_bt
    print(f"Quay lui (O(2^N)): {result_bt}, Thoi gian: {time_bt:.6f}s")
    # --- Quy hoạch động (O(N*T)) ---
    start_dp = time.time()
    result_dp = subset_sum_dp(S, T_final)
    end_dp = time.time()
    time_dp = end_dp - start_dp
    print(f"Quy hoach dong (O(N*T)): {result_dp}, Thoi gian: {time_dp:.6f}s")
    return time_bt, time_dp
# Chạy thử nghiệm:
# Tình huống 1: N nhỏ, T nhỏ (Cả hai đều nhanh)
run_comparison(N=15, T=100)
# Tình huống 2: N lớn (Khoảng 30), T nhỏ (DP thắng tuyệt đối, BT thất bại)
# Cảnh báo: Quay lui có thể mất vài phút hoặc lỗi bộ nhớ nếu N > 40. Ta giới hạn N.
# run_comparison(N=40, T=500) # Chỉ chạy được nếu N < 25
run_comparison(N=20, T=500)
# Tình huống 3: N nhỏ (Khoảng 20), T lớn (DP chậm lại, BT vẫn tệ)
run_comparison(N=20, T=10000)
```

=> Kết quả sau khi chạy chương trình:

```
--- So sanh voi N=16 va Target=50 ---
Quay lui (O(2^N)): True, Thoi gian: 0.000012s
Quy hoach dong (O(N*T)): True, Thoi gian: 0.000044s

--- So sanh voi N=21 va Target=250 ---
Quay lui (O(2^N)): True, Thoi gian: 0.000008s
Quy hoach dong (O(N*T)): True, Thoi gian: 0.000244s

--- So sanh voi N=21 va Target=5000 ---
Quay lui (O(2^N)): True, Thoi gian: 0.000090s
Quy hoach dong (O(N*T)): True, Thoi gian: 0.008115s
```

2.2.2. Code C++:

```
#include <iostream>             // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>        // Thư viện tổng hợp
#include <chrono>               // Đo thời gian chạy Code
#include <random>               // Tạo số ngẫu nhiên
using namespace std;            // Tiền tố std
using namespace chrono;         // Đo thời gian chạy Code
```

```
bool subset_sum_dp(vector<int> S, int T)
{
    // Su dung Quy hoach dong de kiem tra Subset Sum (toi uu hoa khong gian).
    // Do phuc tap: O(N * T).
    int N = S.size();
    // DP[j] = true neu co the tao ra tong j
    vector<bool> dp(T + 1, false);
    dp[0] = true; // Co the tao ra tong 0 (bang tap rong)
    for (int num : S)
    {
        // Duyet nguoc de dam bao moi num chi duoc su dung mot lan
        for (int j = T; j >= num; j--)
        {
            if (dp[j - num])
            {
                dp[j] = true;
            }
        }
    }
    return dp[T];
}
bool subset_sum_backtracking_helper(vector<int>& S, int N, int index, int
current_target)
{
    if (current_target == 0)
        return true;
    if (index == N || current_target < 0)
        return false;
    // Cat tia: Neu phan tu hien tai qua lon, bo qua cac phan tu nho hon
    if (S[index] > current_target)
    {
        return subset_sum_backtracking_helper(S, N, index + 1, current_target);
    }
    // Lua chon 1: Bao gom S[index]
    if (subset_sum_backtracking_helper(S, N, index + 1, current_target - S[index]))
        return true;
    // Lua chon 2: Loai tru S[index]
    return subset_sum_backtracking_helper(S, N, index + 1, current_target);
}
bool subset_sum_backtracking(vector<int> S, int T)
{
    // Su dung thuat toan Quay lui de kiem tra Subset Sum.
    // Do phuc tap: O(2^N).
    int N = S.size();
    // Sap xep giam dan de ho tro cat tia (pruning)
    sort(S.begin(), S.end(), greater<int>());
    return subset_sum_backtracking_helper(S, N, 0, T);
}
void run_comparison(int N, int T)
{
    // Tao tap du lieu: Cac so ngau nhien sao cho tong co the dat duoc T
    random_device rd;
    mt19937 gen(rd());
    uniform_int_distribution<> dist(1, T / N + 1);
    vector<int> S;
    for (int i = 0; i < N; i++)
    {
        S.push_back(dist(gen));
```

```
    }
    S.push_back(T - accumulate(S.begin(), S.end(), 0)); // Chen them phan tu de
tong luon la T
    // Truong hop T lon (T >> 2^N)
    if (N <= 25 && T > 1000)
    {
        // Su dung S da tao
    }
    // Truong hop N lon (2^N >> T)
    else if (N > 25 && T <= 500)
    {
        // Neu N qua lon, Quay lui se that bai nhanh
        S.clear();
        for (int i = 0; i < 25; i++)
        {
            S.push_back(dist(gen));
        }
        N = 25;
        S.push_back(T - accumulate(S.begin(), S.end(), 0));
    }
    // Dam bao S khong chua so am
    vector<int> filtered;
    for (int x : S)
        if (x > 0) filtered.push_back(x);
    S = filtered;
    int T_final = accumulate(S.begin(), S.end(), 0) / 2; // Target cho bai toan Can
Partition
    cout << "\n--- So sanh voi N=" << S.size()
         << " va Target=" << T_final << " ---\n";
    // --- Quay lui (O(2^N)) ---
    auto start_bt = high_resolution_clock::now();
    bool result_bt = subset_sum_backtracking(S, T_final);
    auto end_bt = high_resolution_clock::now();
    double time_bt = duration<double>(end_bt - start_bt).count();
    cout << "Quay lui (O(2^N)): " << result_bt
         << ", Thoi gian: " << time_bt << "s\n";
    // --- Quy hoach dong (O(N*T)) ---
    auto start_dp = high_resolution_clock::now();
    bool result_dp = subset_sum_dp(S, T_final);
    auto end_dp = high_resolution_clock::now();
    double time_dp = duration<double>(end_dp - start_dp).count();
    cout << "Quy hoach dong (O(N*T)): " << result_dp
         << ", Thoi gian: " << time_dp << "s\n";
}
int main()
{
    // Tinh huong 1: N nho, T nho (Ca hai deu nhanh)
    run_comparison(15, 100);
    // Tinh huong 2: N lon, T nho
    run_comparison(20, 500);
    // Tinh huong 3: N nho, T lon
    run_comparison(20, 10000);
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
--- So sanh voi N=16 va Target=50 ---
Quay lui (O(2^N)): 1, Thoi gian: 7.7e-06s
Quy hoach dong (O(N*T)): 1, Thoi gian: 8.45e-05s

--- So sanh voi N=21 va Target=250 ---
Quay lui (O(2^N)): 0, Thoi gian: 0.0217531s
Quy hoach dong (O(N*T)): 0, Thoi gian: 0.0003405s

--- So sanh voi N=21 va Target=5000 ---
Quay lui (O(2^N)): 0, Thoi gian: 0.0232977s
```

2.2.3. Bảng Tổng kết So sánh hiệu suất:

Tiêu chí	Quay lui (Backtracking)	Quy hoạch động (Dynamic Programming - DP)
Độ phức tạp	$O(2^N)$ (Hàm mũ)	O(N . T) (Đa thức giả)
Hiệu suất khi N nhỏ	Tốt.	Tốt.
Hiệu suất khi N lớn (N > 40)	**Rất kém/Không khả thi.** Thời gian tăng theo cấp số nhân 2^N.	**Tốt.** Tăng tuyến tính theo N.
Hiệu suất khi T lớn	Không bị ảnh hưởng trực tiếp bởi T.	**Kém.** Thời gian tăng tuyến tính theo T.
Lựa chọn tối ưu	Khi N rất nhỏ (thường N ≤ 20-25).	Khi N lớn và T nhỏ/vừa phải.
Bản chất	Tìm kiếm tất cả các khả năng.	Xây dựng lời giải từ các bài toán con nhỏ hơn.

=> Trong bài toán **Subset Sum**, **DP** là giải pháp phổ biến và nhanh hơn cho hầu hết các ràng buộc phổ biến (ví dụ: **N ≤ 100** và **T ≤ 10000**). Thuật toán Quay lui chỉ được sử dụng khi **N** rất nhỏ hoặc khi cần thêm các ràng buộc phức tạp không thể mô hình hóa bằng **DP**.

CHỦ ĐỀ 11. THAM LAM & CHIA ĐỂ TRỊ

BÀI 31. THUẬT TOÁN THAM LAM: BÀI TOÁN CHỌN VIỆC

I. Mục Tiêu:

- **Kiến thức:** HS hiểu rõ nguyên tắc hoạt động và điều kiện áp dụng của thuật toán Tham lam. Nắm vững cách chứng minh tính tối ưu của chiến lược Tham lam trong bài toán Chọn việc.
- **Kỹ năng:** HS cài đặt được thuật toán Tham lam để tìm tập hợp hoạt động lớn nhất không trùng lặp (tương thích) trong Python và C++.

- **Tư duy:** HS rèn luyện tư duy lựa chọn tối ưu cục bộ để đạt được tối ưu toàn cục.

II. Phân Bổ Thời Gian (90 Phút):

Hoạt Động	Nội Dung	Thời gian (Phút)
Ôn tập & Giới thiệu	Định nghĩa Thuật toán Tham lam, Giới thiệu Bài toán Chọn việc	15
Lý thuyết	Chiến lược Tham lam (Greedy Choice Property) và Cấu trúc tối ưu con	25
Thực hành	Bài 1. Cài đặt Bài toán Chọn việc O(N log N) (Python & C++)	40
Tổng kết	So sánh với DP, Bài tập về nhà	10
Tổng cộng		**90**

III. Lý Thuyết 1: Giới thiệu Tham lam & Bài toán Chọn việc (15 phút)

1. Thuật toán Tham lam (Greedy Algorithm):

- **Nguyên tắc:** Tại mỗi bước ra quyết định, thuật toán Tham lam luôn chọn giải pháp **tối ưu cục bộ** (best local choice) với hy vọng rằng chuỗi các lựa chọn tối ưu cục bộ sẽ dẫn đến giải pháp **tối ưu toàn cục** (global optimum).
- **Điều kiện áp dụng:**
  + **Tính chất lựa chọn Tham lam (Greedy Choice Property):** Có một lựa chọn tối ưu toàn cục bắt đầu bằng lựa chọn Tham lam đầu tiên.
  + **Cấu trúc tối ưu con (Optimal Substructure):** Giải pháp tối ưu cho bài toán lớn chứa các giải pháp tối ưu cho các bài toán con.

2. Bài toán Chọn việc (Activity Selection Problem):

- **Đầu vào:** Một tập hợp **S = {a_1, a_2, …, a_n}** các hoạt động (Ví dụ: các cuộc họp, các lớp học). Mỗi hoạt động **a_i** có thời gian bắt đầu **s_i** và thời gian kết thúc **f_i (s_i < f_i)**.
- **Ràng buộc:** Hai hoạt động **a_i** và **a_j** được gọi là **tương thích** (compatible) nếu chúng không chồng chéo về thời gian (tức là **f_i ≤ s_j** hoặc **f_j ≤ s_i**).
- **Mục tiêu:** Tìm tập hợp con lớn nhất các hoạt động tương thích.

IV. Lý thuyết 2: Chiến lược Tham lam cho Bài toán Chọn việc (25 phút)

1. Các chiến lược Tham lam có thể có:

- **Chọn hoạt động ngắn nhất:** (Sai => Có thể bỏ lỡ hai hoạt động dài hơn).
- **Chọn hoạt động bắt đầu sớm nhất:** (Sai => Hoạt động này có thể kết thúc rất muộn).
- **Chọn hoạt động kết thúc sớm nhất (The Optimal Greedy Choice):** (Đúng)

2. Chứng minh Chiến lược Tối ưu:

- **Chiến lược Tham lam Tối ưu:** Luôn chọn hoạt động **kết thúc sớm nhất** trong số các hoạt động tương thích còn lại.
- **Lý do:** Bằng cách chọn hoạt động **a_k** kết thúc sớm nhất, ta **giải phóng khoảng thời gian lớn nhất có thể** sau nó để dành cho các hoạt động tiếp theo.

3. Công thức Thuật toán:

- **Sắp xếp:** Sắp xếp tất cả các hoạt động theo thứ tự **tăng dần** của **thời gian kết thúc (f_i)**.

- Lựa chọn Tham lam:

+ Chọn hoạt động đầu tiên (hoạt động kết thúc sớm nhất).

+ Duyệt qua các hoạt động còn lại: Chỉ chọn hoạt động **a_i** nếu thời gian bắt đầu **s_i** của nó **lớn hơn hoặc bằng** thời gian kết thúc **f_last** của hoạt động vừa chọn.

V. Thực Hành: Cài đặt Bài toán Chọn việc (40 phút)

Bài toán: Cho danh sách **N** hoạt động, tìm số lượng hoạt động tối đa có thể thực hiện.

1. Code Python:

```
def activity_selection(activities):
    # Giải bài toán Chọn việc bằng thuật toán Tham lam.
    # Chiến lược tối ưu: Luôn chọn hoạt động kết thúc sớm nhất.
    # Tham số:
    # activities (list): Danh sách các hoạt động, mỗi hoạt động là
    # một tuple/list: (start_time, finish_time).
    # Trả về:
    # tuple: (Số lượng hoạt động tối đa được chọn, Danh sách các hoạt động đã chọn)
    # 1. KIỂM TRA ĐẦU VÀO
    if not activities:
        return 0, []
    # 2. SẮP XẾP THAM LAM (Greedy Choice Property)
    # Sắp xếp các hoạt động theo thời gian kết thúc (finish_time, index 1) tăng
dần.
    # Độ phức tạp: O(N log N)
    activities.sort(key=lambda x: x[1])
    # 3. KHỞI TẠO
    count = 0
    selected = []
    last_finish_time = -1  # Sử dụng -1 hoặc thời gian kết thúc của hoạt động giả
định trước
                           # để đảm bảo hoạt động đầu tiên luôn được chọn.
    # 4. DUYỆT VÀ LỰA CHỌN THAM LAM (O(N))
    for start_time, current_finish_time in activities:
        # Lựa chọn Tham lam: Kiểm tra sự tương thích
        # Nếu thời gian bắt đầu của hoạt động hiện tại (start_time)
        # lớn hơn hoặc bằng thời gian kết thúc của hoạt động vừa chọn
(last_finish_time),
        # thì hoạt động này tương thích và là lựa chọn tối ưu cục bộ.
        if start_time >= last_finish_time:
            count += 1
            selected.append((start_time, current_finish_time))
            # Cập nhật: Cài đặt lại rào chắn thời gian kết thúc
            last_finish_time = current_finish_time
    return count, selected
# ----------------- CHẠY THỬ VÀ MINH HỌA -----------------
S = [ (1, 4), (3, 5), (0, 6), (5, 7), (8, 9), (5, 9), (2, 14), (12, 16) ]
max_count, selected_activities = activity_selection(S)
print(f"Hoat dong ban dau (start, finish): {S}")
print("-" * 50)
print(f"So luong hoat dong toi da duoc chon: {max_count}")
print(f"Cac hoat dong duoc chon:")
for activity in selected_activities:
    print(f"  {activity}")
# Minh họa quá trình chọn (sau khi sắp xếp theo finish time):
```

```
# 1. activities được sắp xếp: [(1, 4), (3, 5), (0, 6), (5, 7), (8, 9), (5, 9), (2,
14), (12, 16)]
# 2. Chọn (1, 4). last_finish_time = 4.
# 3. Xét (3, 5). 3 < 4. Bỏ qua.
# 4. Xét (0, 6). 0 < 4. Bỏ qua.
# 5. Xét (5, 7). 5 >= 4. Chọn. last_finish_time = 7.
# 6. Xét (8, 9). 8 >= 7. Chọn. last_finish_time = 9.
# 7. Xét (5, 9). 5 < 9. Bỏ qua.
# 8. Xét (2, 14). 2 < 9. Bỏ qua.
# 9. Xét (12, 16). 12 >= 9. Chọn. last_finish_time = 16.
# Kết quả: 4 hoạt động.
```

=> Kết quả sau khi chạy chương trình:

```
Hoat dong ban dau (start, finish): [(1, 4), (3, 5), (0, 6), (5, 7), (8, 9), (5, 9), (2, 14), (12, 16)]
--------------------------------------------------
So luong hoat dong toi da duoc chon: 4
Cac hoat dong duoc chon:
  (1, 4)
  (5, 7)
  (8, 9)
  (12, 16)
```

- Giải thích Code Python:

+ Sắp xếp theo key: Python cho phép sắp xếp theo khóa là thời gian kết thúc (**lambda x: x[1]**).

+ **Lựa chọn Tham lam:** Điều kiện **start_time >= last_finish_time** là bước ra quyết định quan trọng nhất. Nếu tương thích, ta cập nhật **last_finish_time** bằng thời gian kết thúc của hoạt động mới.

2. Code C++:

```
#include <iostream>             // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>        // Thư viện tổng hợp
using namespace std;            // Tiền tố std
pair<int, vector<pair<int, int>>> activity_selection(vector<pair<int, int>>
activities)
{
    // Giai bai toan Chon viec bang thuat toan Tham lam.
    // Chien luoc toi uu: Luon chon hoat dong ket thuc som nhat.
    // Tham so:
    // activities (vector): Danh sach cac hoat dong, moi hoat dong la
    // mot cap (start_time, finish_time).
    // Tra ve:
    // pair: (So luong hoat dong toi da duoc chon, Danh sach cac hoat dong da chon)
    // 1. KIEM TRA DAU VAO
    if (activities.empty())
    {
        return {0, {}};
    }
    // 2. SAP XEP THAM LAM (Greedy Choice Property)
    // Sap xep cac hoat dong theo thoi gian ket thuc (finish_time) tang dan.
    // Do phuc tap: O(N log N)
    sort(activities.begin(), activities.end(),
         [](const pair<int, int>& a, const pair<int, int>& b)
         {
             return a.second < b.second;
         });
    // 3. KHOI TAO
```

```
    int count = 0;
    vector<pair<int, int>> selected;
    int last_finish_time = -1; // Su dung -1 de dam bao hoat dong dau tien luon
duoc chon
    // 4. DUYET VA LUA CHON THAM LAM (O(N))
    for (auto activity : activities)
    {
        int start_time = activity.first;
        int current_finish_time = activity.second;
        // Lua chon Tham lam: Kiem tra su tuong thich
        // Neu start_time >= last_finish_time thi hoat dong tuong thich
        if (start_time >= last_finish_time)
        {
            count++;
            selected.push_back(activity);
            // Cap nhat thoi gian ket thuc gan nhat
            last_finish_time = current_finish_time;
        }
    }
    return {count, selected};
}
int main()
{
    // ----------------- CHAY THU VA MINH HOA -----------------
    vector<pair<int, int>> S = {
        {1, 4}, {3, 5}, {0, 6}, {5, 7},
        {8, 9}, {5, 9}, {2, 14}, {12, 16}
    };
    auto result = activity_selection(S);
    int max_count = result.first;
    vector<pair<int, int>> selected_activities = result.second;
    cout << "Hoat dong ban dau (start, finish):\n";
    for (auto a : S)
    {
        cout << "(" << a.first << ", " << a.second << ") ";
    }
    cout << "\n--------------------------------------------------\n";
    cout << "So luong hoat dong toi da duoc chon: " << max_count << "\n";
    cout << "Cac hoat dong duoc chon:\n";
    for (auto activity : selected_activities)
    {
        cout << "  (" << activity.first << ", " << activity.second << ")\n";
    }
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
Hoat dong ban dau (start, finish):
(1, 4) (3, 5) (0, 6) (5, 7) (8, 9) (5, 9) (2, 14) (12, 16)
--------------------------------------------------
So luong hoat dong toi da duoc chon: 4
Cac hoat dong duoc chon:
  (1, 4)
  (5, 7)
  (8, 9)
  (12, 16)
```

VI. Tổng kết & Bài tập Về nhà (10 phút)

1. Tóm tắt Nguyên lý Tham lam (Bài toán Chọn việc):

- **Chiến lược:** Sắp xếp tất cả các hoạt động theo thứ tự **thời gian kết thúc** tăng dần.
- **Lựa chọn:** Luôn chọn hoạt động **kết thúc sớm nhất** trong số các hoạt động **tương thích** còn lại.
- **Độ phức tạp:** O(N log N) (chủ yếu do bước sắp xếp).

2. Bài tập Về nhà:

2.1. Bài tập 1 (Phòng họp Tối thiểu): Cho một danh sách các cuộc họp, tìm số lượng phòng họp tối thiểu cần thiết để chứa tất cả các cuộc họp (Tức là tìm số lượng cuộc họp chồng chéo lớn nhất tại bất kỳ thời điểm nào). (Gợi ý: Sắp xếp theo thời gian bắt đầu và dùng **Min-Heap**).

2.1.1. Code Python:

```
import heapq    # Min-Heap
def min_meeting_rooms(meetings):
    # Tìm số lượng phòng họp tối thiểu cần thiết cho tất cả các cuộc họp.
    # Tham số:
    # meetings (list): Danh sách các cuộc họp, mỗi cuộc họp là [start_time,
end_time].
    # Trả về:
    # int: Số phòng họp tối thiểu.
    if not meetings:
        return 0
    # 1. Sắp xếp Tham lam: Sắp xếp tất cả các cuộc họp theo thời gian BẮT ĐẦU tăng
dần (O(N log N))
    meetings.sort(key=lambda x: x[0])
    # 2. Khởi tạo Min-Heap
    # Heap này chỉ lưu trữ thời gian KẾT THÚC của các phòng họp đang được sử dụng.
    # Phần tử nhỏ nhất (top) là phòng kết thúc sớm nhất.
    # Trong Python, heapq là Min-Heap.
    meeting_ends = []
    # Bắt đầu với cuộc họp đầu tiên: Cần 1 phòng
    # Thêm thời gian kết thúc của cuộc họp đầu tiên vào Heap
    heapq.heappush(meeting_ends, meetings[0][1])
    # 3. Duyệt và Ra quyết định Tham lam (O(N log N))
    for i in range(1, len(meetings)):
        current_start = meetings[i][0]
        current_end = meetings[i][1]
        # Lấy thời gian kết thúc sớm nhất của phòng họp đang được sử dụng (Min-
Heap.top())
        earliest_finish = meeting_ends[0]
        # A. Quyết định Tham lam: SỬ DỤNG LẠI PHÒNG?
        # Nếu cuộc họp mới bắt đầu sau hoặc đúng lúc phòng kết thúc sớm nhất,
        # ta có thể sử dụng lại phòng đó.
        if current_start >= earliest_finish:
            # Loại bỏ thời gian kết thúc cũ (phòng đã được giải phóng)
            heapq.heappop(meeting_ends)
        # B. DÙNG PHÒNG MỚI HOẶC PHÒNG ĐƯỢC TÁI CHẾ
        # Thêm thời gian kết thúc của cuộc họp hiện tại vào Heap.
        # Nếu ta tái chế phòng (trường hợp A), ta thay thế thời gian kết thúc.
        # Nếu ta cần phòng mới (trường hợp B), ta thêm một phần tử mới.
        heapq.heappush(meeting_ends, current_end)
    # 4. Kết quả
    # Kích thước cuối cùng của Heap là số phòng cần thiết
```

```
    # (vì nó đại diện cho số lượng cuộc họp chồng chéo lớn nhất).
    return len(meeting_ends)
# ---------------- CHẠY THỬ VÀ MINH HỌA ----------------
# [1, 5], [2, 6], [3, 7] chồng chéo nhau. (3 phòng)
# Ví dụ 1: Cần 3 phòng.
M1 = [[0, 30], [5, 10], [15, 20]]
# Sắp xếp: [[0, 30], [5, 10], [15, 20]]
# 1. [0, 30]. Heap: [30]. Rooms: 1.
# 2. [5, 10]. 5 < 30. Heap: [10, 30]. Rooms: 2.
# 3. [15, 20]. 15 > 10. Pop 10. Heap: [30]. Push 20. Heap: [20, 30]. Rooms: 2.
print(f"Meetings 1: {M1} -> Phong hop toi thieu: {min_meeting_rooms(M1)}") # Kết
quả: 2
# Ví dụ 2: Cần 3 phòng.
M2 = [[1, 18], [18, 23], [15, 29], [4, 15], [2, 11], [5, 13]]
# Sắp xếp: [[1, 18], [2, 11], [4, 15], [5, 13], [15, 29], [18, 23]]
# 1. [1, 18]. Heap: [18]. R=1.
# 2. [2, 11]. 2 < 18. Heap: [11, 18]. R=2.
# 3. [4, 15]. 4 < 11. Heap: [11, 18, 15]. R=3. (Chồng chéo lớn nhất: [1,18],
[2,11], [4,15] tại t=4)
# 4. [5, 13]. 5 < 11. Heap: [11, 18, 15, 13]. R=4.
# 5. [15, 29]. 15 > 11. Pop 11. Push 29. Heap: [13, 18, 15, 29]. R=4.
# ...
print(f"Meetings 2: {M2} -> Phong hop toi thieu: {min_meeting_rooms(M2)}") # Kết
quả: 4
```

=> Kết quả sau khi chạy chương trình:

```
Meetings 1: [[0, 30], [5, 10], [15, 20]] -> Phong hop toi thieu: 2
Meetings 2: [[1, 18], [18, 23], [15, 29], [4, 15], [2, 11], [5, 13]] -> Phong hop toi thieu: 4
```

2.1.2. Code C++:

```
#include <iostream>             // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>        // Thư viện tổng hợp
using namespace std;            // Tiền tố std
int min_meeting_rooms(vector<vector<int>> meetings)
{
    // Tim so luong phong hop toi thieu can thiet cho tat ca cac cuoc hop.
    // Tham so:
    // meetings (vector): Danh sach cac cuoc hop, moi cuoc hop la [start_time,
end_time].
    // Tra ve:
    // int: So phong hop toi thieu.
    if (meetings.empty())
    {
        return 0;
    }
    // 1. Sap xep Tham lam:
    // Sap xep cac cuoc hop theo thoi gian BAT DAU tang dan (O(N log N))
    sort(meetings.begin(), meetings.end(),
         [](const vector<int>& a, const vector<int>& b)
         {
             return a[0] < b[0];
         });
    // 2. Khoi tao Min-Heap
    // Heap chi luu tru thoi gian KET THUC cua cac phong hop dang duoc su dung.
    // Phan tu nho nhat la phong ket thuc som nhat.
    priority_queue<int, vector<int>, greater<int>> meeting_ends;
    // Bat dau voi cuoc hop dau tien: Can 1 phong
    meeting_ends.push(meetings[0][1]);
    // 3. Duyet va Ra quyet dinh Tham lam (O(N log N))
```

```
    for (int i = 1; i < meetings.size(); i++)
    {
        int current_start = meetings[i][0];
        int current_end = meetings[i][1];
        // Lay thoi gian ket thuc som nhat cua phong hop dang su dung
        int earliest_finish = meeting_ends.top();
        // A. Quyet dinh Tham lam: SU DUNG LAI PHONG?
        if (current_start >= earliest_finish)
        {
            // Phong cu da duoc giai phong
            meeting_ends.pop();
        }
        // B. DUNG PHONG MOI HOAC PHONG DUOC TAI CHE
        // Them thoi gian ket thuc cua cuoc hop hien tai vao Heap
        meeting_ends.push(current_end);
    }
    // 4. Ket qua
    // Kich thuoc cua Heap la so phong can thiet
    return meeting_ends.size();
}
int main()
{
    // ----------------- CHAY THU VA MINH HOA -----------------
    // Vi du 1
    vector<vector<int>> M1 = {{0, 30}, {5, 10}, {15, 20}};
    cout << "Meetings 1: ";
    for (auto m : M1)
    {
        cout << "[" << m[0] << ", " << m[1] << "] ";
    }
    cout << "-> Phong hop toi thieu: "
         << min_meeting_rooms(M1) << endl;
    // Vi du 2
    vector<vector<int>> M2 = {
        {1, 18}, {18, 23}, {15, 29},
        {4, 15}, {2, 11}, {5, 13}
    };
    cout << "Meetings 2: ";
    for (auto m : M2)
    {
        cout << "[" << m[0] << ", " << m[1] << "] ";
    }
    cout << "-> Phong hop toi thieu: "
         << min_meeting_rooms(M2) << endl;
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
Meetings 1: [0, 30] [5, 10] [15, 20] -> Phong hop toi thieu: 2
Meetings 2: [1, 18] [18, 23] [15, 29] [4, 15] [2, 11] [5, 13] -> Phong hop toi thieu: 4
```

2.2. Bài tập 2 (Tham lam Đổi tiền): Cho một số tiền **V**, tìm số lượng đồng xu tối thiểu cần thiết để đổi được số tiền đó, với một tập hợp các mệnh giá tiền nhất định (ví dụ: {1, 2, 5, 10, 20, 50}). Kiểm tra xem chiến lược Tham lam luôn tối ưu cho bài toán đổi tiền không.

2.2.1. Code Python:

```
def greedy_coin_change(denominations, V):
    # Giải bài toán Đổi tiền bằng thuật toán Tham lam.
```

```
    # Tham số:
    # denominations (list): Danh sách các mệnh giá đồng xu.
    # V (int): Số tiền cần đổi.
    # Trả về:
    # tuple: (Số lượng đồng xu tối thiểu tìm được, Chi tiết đổi tiền)
    # 1. Chiến lược Tham lam: Sắp xếp giảm dần để luôn chọn mệnh giá lớn nhất trước
    denominations.sort(reverse=True)
    count = 0
    change_details = {}
    remaining_amount = V
    print(f"\n--- Thu ap dung Tham lam cho so tien V = {V} ---")
    for coin in denominations:
        if remaining_amount == 0:
            break
        # 2. Lựa chọn Tham lam: Số lượng đồng xu lớn nhất có thể chọn
        num_coins = remaining_amount // coin
        if num_coins > 0:
            count += num_coins
            change_details[coin] = num_coins
            remaining_amount -= num_coins * coin
            print(f"  Chon {num_coins} dong xu menh gia {coin}. Con lai:
{remaining_amount}")
    return count, change_details
def check_optimality():
    # Kiểm tra xem thuật toán Tham lam có luôn tối ưu cho bài toán Đổi tiền
không.
    print("\n==========================================")
    print("3. KIEM TRA TINH TOI UU CUA THUAT TOAN THAM LAM")
    # Mệnh giá tiền tệ tiêu chuẩn (USD, VND, Euro,...)
    STANDARD_COINS = [50, 20, 10, 5, 2, 1]
    # Mệnh giá tiền tệ KHÔNG TIÊU CHUẨN (trường hợp Tham lam thất bại)
    NON_STANDARD_COINS = [1, 5, 6, 9]
    V_FAILURE = 11
    # --- Trường hợp 1: Mệnh giá tiêu chuẩn (Tham lam Tối ưu) ---
    V1 = 128
    count1, details1 = greedy_coin_change(STANDARD_COINS, V1)
    print(f"\n[Truong hop 1 - Tieu chuan: {STANDARD_COINS}]")
    print(f"Tong so dong xu can dung cho {V1}: {count1}")
    print(f"Chi tiet: {details1}")
    print("-> Ket qua: Tham lam TOI UU trong he thong tien te tieu chuan.")
    # --- Trường hợp 2: Mệnh giá không tiêu chuẩn (Tham lam Thất bại) ---
    count2, details2 = greedy_coin_change(NON_STANDARD_COINS, V_FAILURE)
    # Phân tích kết quả Tham lam:
    # Tham lam chọn [9, 1, 1] -> 3 đồng xu.
    # Lời giải tối ưu: [5, 6] -> 2 đồng xu.
    print(f"\n[Truong hop 2 - KHONG Tieu chuan: {NON_STANDARD_COINS}]")
    print(f"Tong so dong xu Tham lam tim duoc cho {V_FAILURE}: {count2}")
    print(f"Chi tiet: {details2}")
    print("\nLoi giai toi uu thuc te (DP/Thu cong): [5, 6] -> Chi can 2 dong xu.")
    print("-> Ket qua: Tham lam KHONG TOI UU trong he thong tien te nay.")
# ----------------- CHẠY THỬ VÀ SO SÁNH -----------------
check_optimality()
```

=> Kết quả sau khi chạy chương trình:

```
3. KIEM TRA TINH TOI UU CUA THUAT TOAN THAM LAM

--- Thu ap dung Tham lam cho so tien V = 128 ---
  Chon 2 dong xu menh gia 50. Con lai: 28
  Chon 1 dong xu menh gia 20. Con lai: 8
  Chon 1 dong xu menh gia 5. Con lai: 3
  Chon 1 dong xu menh gia 2. Con lai: 1
  Chon 1 dong xu menh gia 1. Con lai: 0

[Truong hop 1 - Tieu chuan: [50, 20, 10, 5, 2, 1]]
Tong so dong xu can dung cho 128: 6
Chi tiet: {50: 2, 20: 1, 5: 1, 2: 1, 1: 1}
-> Ket qua: Tham lam TOI UU trong he thong tien te tieu chuan.

--- Thu ap dung Tham lam cho so tien V = 11 ---
  Chon 1 dong xu menh gia 9. Con lai: 2
  Chon 2 dong xu menh gia 1. Con lai: 0

[Truong hop 2 - KHONG Tieu chuan: [9, 6, 5, 1]]
Tong so dong xu Tham lam tim duoc cho 11: 3
Chi tiet: {9: 1, 1: 2}

Loi giai toi uu thuc te (DP/Thu cong): [5, 6] -> Chi can 2 dong xu.
-> Ket qua: Tham lam KHONG TOI UU trong he thong tien te nay.
```

2.2.2. Code C++:

```cpp
#include <iostream>             // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>        // Thư viện tổng hợp
using namespace std;            // Tiền tố std
// Ham giai bai toan Doi tien bang thuat toan Tham lam
pair<int, map<int, int>> greedy_coin_change(vector<int> denominations, int V)
{
    // Giai bai toan Doi tien bang thuat toan Tham lam.
    // Tham so:
    // denominations (vector): Danh sach cac menh gia dong xu.
    // V (int): So tien can doi.
    // Tra ve:
    // pair: (So luong dong xu toi thieu tim duoc, Chi tiet doi tien)
    // 1. Chien luoc Tham lam: Sap xep giam dan de luon chon menh gia lon nhat
truoc
    sort(denominations.begin(), denominations.end(), greater<int>());
    int count = 0;
    map<int, int> change_details;
    int remaining_amount = V;
    cout << "\n--- Thu ap dung Tham lam cho so tien V = " << V << " ---" << endl;
    for (int coin : denominations)
    {
        if (remaining_amount == 0)
            break;
        // 2. Lua chon Tham lam: So luong dong xu lon nhat co the chon
        int num_coins = remaining_amount / coin;
        if (num_coins > 0)
        {
            count += num_coins;
            change_details[coin] = num_coins;
            remaining_amount -= num_coins * coin;
```

```
            cout << "  Chon " << num_coins
                 << " dong xu menh gia " << coin
                 << ". Con lai: " << remaining_amount << endl;
        }
    }
    return {count, change_details};
}
// Ham kiem tra tinh toi uu cua thuat toan Tham lam
void check_optimality() {
    // Kiem tra xem thuat toan Tham lam co luon toi uu cho bai toan Doi tien khong.
    cout << "\n========================================" << endl;
    cout << "3. KIEM TRA TINH TOI UU CUA THUAT TOAN THAM LAM" << endl;
    // Menh gia tien te tieu chuan (USD, VND, Euro,...)
    vector<int> STANDARD_COINS = {50, 20, 10, 5, 2, 1};
    // Menh gia tien te KHONG TIEU CHUAN (truong hop Tham lam that bai)
    vector<int> NON_STANDARD_COINS = {1, 5, 6, 9};
    int V_FAILURE = 11;
    // --- Truong hop 1: Menh gia tieu chuan (Tham lam Toi uu) ---
    int V1 = 128;
    auto result1 = greedy_coin_change(STANDARD_COINS, V1);
    cout << "\n[Truong hop 1 - Tieu chuan: ";
    for (int c : STANDARD_COINS) cout << c << " ";
    cout << "]" << endl;
    cout << "Tong so dong xu can dung cho " << V1 << ": "
         << result1.first << endl;
    cout << "Chi tiet: ";
    for (auto &p : result1.second)
        cout << "(" << p.first << ":" << p.second << ") ";
    cout << endl;
    cout << "-> Ket qua: Tham lam TOI UU trong he thong tien te tieu chuan." <<
endl;
    // --- Truong hop 2: Menh gia khong tieu chuan (Tham lam That bai) ---
    auto result2 = greedy_coin_change(NON_STANDARD_COINS, V_FAILURE);
    cout << "\n[Truong hop 2 - KHONG Tieu chuan: ";
    for (int c : NON_STANDARD_COINS) cout << c << " ";
    cout << "]" << endl;
    cout << "Tong so dong xu Tham lam tim duoc cho "
         << V_FAILURE << ": " << result2.first << endl;
    cout << "Chi tiet: ";
    for (auto &p : result2.second)
        cout << "(" << p.first << ":" << p.second << ") ";
    cout << endl;
    cout << "\nLoi giai toi uu thuc te (DP/Thu cong): [5, 6] -> Chi can 2 dong xu."
<< endl;
    cout << "-> Ket qua: Tham lam KHONG TOI UU trong he thong tien te nay." <<
endl;
}
// ----------------- CHAY THU VA SO SANH -----------------
int main()
{
    check_optimality();
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
3. KIEM TRA TINH TOI UU CUA THUAT TOAN THAM LAM

--- Thu ap dung Tham lam cho so tien V = 128 ---
  Chon 2 dong xu menh gia 50. Con lai: 28
  Chon 1 dong xu menh gia 20. Con lai: 8
  Chon 1 dong xu menh gia 5. Con lai: 3
  Chon 1 dong xu menh gia 2. Con lai: 1
  Chon 1 dong xu menh gia 1. Con lai: 0

[Truong hop 1 - Tieu chuan: 50 20 10 5 2 1 ]
Tong so dong xu can dung cho 128: 6
Chi tiet: (1:1) (2:1) (5:1) (20:1) (50:2)
-> Ket qua: Tham lam TOI UU trong he thong tien te tieu chuan.

--- Thu ap dung Tham lam cho so tien V = 11 ---
  Chon 1 dong xu menh gia 9. Con lai: 2
  Chon 2 dong xu menh gia 1. Con lai: 0

[Truong hop 2 - KHONG Tieu chuan: 1 5 6 9 ]
Tong so dong xu Tham lam tim duoc cho 11: 3
Chi tiet: (1:2) (9:1)

Loi giai toi uu thuc te (DP/Thu cong): [5, 6] -> Chi can 2 dong xu.
-> Ket qua: Tham lam KHONG TOI UU trong he thong tien te nay.
```

BÀI 32. THUẬT TOÁN CHIA ĐỂ TRỊ - SẮP XẾP TRỘN (MERGE SORT)

I. Mục Tiêu:

- **Kiến thức:** HS hiểu rõ nguyên tắc hoạt động của phương pháp **Chia để trị** (Divide and Conquer). Nắm vững cấu trúc và các bước của thuật toán Merge Sort.
- **Kỹ năng:** HS cài đặt được thuật toán Merge Sort O(N log N) sử dụng đệ quy trong Python và C++.
- **Tư duy:** HS phân tích được độ phức tạp thời gian và không gian của Merge Sort thông qua phương pháp đệ quy và hiểu sự khác biệt so với các thuật toán $O(N^2)$.

II. Phân Bổ Thời Gian (90 Phút):

Hoạt Động	Nội Dung	Thời gian (Phút)
Ôn tập & Giới thiệu	Định nghĩa Chia để trị, Giới thiệu Merge Sort	15
Lý thuyết	Cấu trúc 3 bước: Chia, Trị, Kết hợp (Merge)	25
Thực hành	Bài 1. Cài đặt hàm Merge và hàm MergeSort (Python & C++)	40
Tổng kết	Độ phức tạp, Ưu/Nhược điểm, Bài tập về nhà	10
Tổng cộng		**90**

III. Lý Thuyết 1: Giới thiệu Chia để trị & Merge Sort (15 phút)

1. Phương pháp Chia để trị (Divide and Conquer):

Đây là một khuôn mẫu thuật toán mạnh mẽ, giải quyết một bài toán lớn bằng ba bước sau:

- **Chia (Divide):** Chia bài toán lớn thành nhiều bài toán con nhỏ hơn, độc lập. Các bài toán con này tương tự bài toán gốc nhưng có kích thước nhỏ hơn.
- **Trị (Conquer):** Giải quyết đệ quy các bài toán con (hoặc giải quyết trực tiếp nếu bài toán con đủ nhỏ, đạt điều kiện dừng).
- **Kết hợp (Combine):** Kết hợp các lời giải của các bài toán con để tạo thành lời giải cho bài toán ban đầu.

2. Giới thiệu Merge Sort (Sắp xếp Trộn):

- Merge Sort là một thuật toán sắp xếp dựa trên nguyên tắc Chia để trị.
- **Ý tưởng chính:** Một danh sách **N** phần tử được chia đôi liên tục cho đến khi chỉ còn các danh sách một phần tử (đã được sắp xếp). Sau đó, các danh sách đã sắp xếp này được **trộn (Merge)** lại với nhau theo thứ tự tăng dần.

IV. Lý thuyết 2: Cấu trúc 3 bước của Merge Sort (25 phút)

Ta xét một mảng **A** từ chỉ mục **p** đến **r**.

1. Bước Chia (Divide):

- Tìm điểm giữa của mảng: q = (p + r)/2.
- Chia mảng A[p..r] thành hai mảng con: A[p..q] và A[q+1..r].
- Bước này có độ phức tạp O(1).

2. Bước Trị (Conquer):

- Gọi đệ quy Merge Sort cho mảng con bên trái: MergeSort(A, p, q).
- Gọi đệ quy Merge Sort cho mảng con bên phải: MergeSort(A, q+1, r).
- **Điều kiện dừng (Base Case):** Khi mảng con chỉ còn một phần tử (p ≥ r), nó đã được sắp xếp.

3. Bước Kết hợp/Trộn (Combine/Merge):

- Đây là bước quan trọng nhất và tốn thời gian nhất. Ta trộn hai mảng con A[p..q] và A[q+1..r] (vốn đã được sắp xếp) thành một mảng lớn hơn đã được sắp xếp.
- **Quá trình trộn:**
  + Tạo hai mảng tạm (temp arrays) chứa hai mảng con.
  + Sử dụng hai con trỏ (pointers), mỗi con trỏ trỏ vào đầu mỗi mảng tạm.
  + Tại mỗi bước, so sánh hai phần tử được trỏ tới và di chuyển phần tử nhỏ hơn vào mảng gốc **A**.
- Bước này có độ phức tạp O(N), vì mỗi phần tử chỉ được di chuyển đúng một lần.

- Độ phức tạp:

+ **Thời gian:** T(N) = 2T(N/2) + O(N) (Chia và Trị => 2T(N/2), Kết hợp => O(N)). Giải phương trình đệ quy này cho ra kết quả: **O(N log N)** (trong mọi trường hợp: Tốt nhất, xấu nhất, trung bình).

+ **Không gian:** O(N) (do phải sử dụng các mảng tạm trong bước trộn).

V. Thực Hành: Cài đặt Merge Sort (40 phút)

1. Code Python:

```python
def merge(left, right):
    # Bước 3: Hàm Kết hợp (Combine) - Trộn hai mảng con đã sắp xếp (left và right)
    # thành một mảng lớn đã sắp xếp duy nhất.
    # Độ phức tạp thời gian: O(N)
    merged = []
    i = j = 0  # i là con trỏ cho mảng left, j là con trỏ cho mảng right
    # So sánh và thêm phần tử nhỏ hơn vào merged
```

```python
    while i < len(left) and j < len(right):
        if left[i] <= right[j]:
            merged.append(left[i])
            i += 1
        else:
            merged.append(right[j])
            j += 1
    # Thêm các phần tử còn lại (nếu một trong hai mảng đã hết trước)
    merged.extend(left[i:])
    merged.extend(right[j:])
    return merged
def merge_sort(arr):
    # Hàm chính thực hiện Merge Sort (Chia và Trị)
    # Độ phức tạp thời gian: O(N log N)
    # Điều kiện Dừng (Base Case): Mảng có 0 hoặc 1 phần tử đã được sắp xếp
    if len(arr) <= 1:
        return arr
    # Bước 1: Chia (Divide)
    mid = len(arr) // 2
    left = arr[:mid]  # Nửa trái
    right = arr[mid:] # Nửa phải
    # Bước 2: Trị (Conquer - Gọi đệ quy)
    # Sắp xếp đệ quy nửa trái và nửa phải
    left_sorted = merge_sort(left)
    right_sorted = merge_sort(right)
    # Bước 3: Kết hợp (Combine)
    # Trộn hai nửa đã sắp xếp lại
    return merge(left_sorted, right_sorted)
# ----------------- CHẠY THỬ VÀ MINH HỌA -----------------
A = [12, 11, 13, 5, 6, 7]
print(f"Mang goc: {A}")
sorted_A = merge_sort(A)
print(f"Mang da sap xep: {sorted_A}")
# Minh họa quá trình Merge Sort:
# [12, 11, 13, 5, 6, 7]
# Chia thành: [12, 11, 13] và [5, 6, 7]
# ... (Đệ quy tiếp tục chia)
# Merge: [11, 12, 13] và [5, 6, 7]
# Merge cuối cùng: [5, 6, 7, 11, 12, 13]
```

=> Kết quả sau khi chạy chương trình:

```
Mang goc: [12, 11, 13, 5, 6, 7]
Mang da sap xep: [5, 6, 7, 11, 12, 13]
```

- Giải thích Code Python:

Cài đặt Python: Merge Sort trong Python thường tạo ra các mảng con mới (tốn O(N) không gian) để đơn giản hóa cú pháp. Hàm **merge** nhận hai mảng đã sắp xếp, dùng hai con trỏ **(i, j)** để so sánh và xây dựng mảng **merged mới**. Đây là cách trực quan nhất để thể hiện bước **Kết hợp**.

2. Code C++:

```cpp
#include <iostream>             // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>        // Thư viện tổng hợp
using namespace std;            // Tiền tố std
// Ham ket hop (Combine) - Tron hai mang con da sap xep
vector<int> merge(const vector<int>& left, const vector<int>& right)
{
    // Buoc 3: Ham Ket hop (Combine) - Tron hai mang con da sap xep (left va right)
```

```
    // thanh mot mang lon da sap xep duy nhat.
    // Do phuc tap thoi gian: O(N)
    vector<int> merged;
    int i = 0, j = 0;  // i la con tro cho mang left, j la con tro cho mang right
    // So sanh va them phan tu nho hon vao merged
    while (i < left.size() && j < right.size())
    {
        if (left[i] <= right[j])
        {
            merged.push_back(left[i]);
            i++;
        }
        else
        {
            merged.push_back(right[j]);
            j++;
        }
    }
    // Them cac phan tu con lai (neu mot trong hai mang da het truoc)
    while (i < left.size())
    {
        merged.push_back(left[i]);
        i++;
    }
    while (j < right.size())
    {
        merged.push_back(right[j]);
        j++;
    }
    return merged;
}
// Ham chinh thuc hien Merge Sort (Chia va Tri)
vector<int> merge_sort(const vector<int>& arr)
{
    // Ham chinh thuc hien Merge Sort (Chia va Tri)
    // Do phuc tap thoi gian: O(N log N)
    // Dieu kien Dung (Base Case): Mang co 0 hoac 1 phan tu da duoc sap xep
    if (arr.size() <= 1)
        return arr;
    // Buoc 1: Chia (Divide)
    int mid = arr.size() / 2;
    vector<int> left(arr.begin(), arr.begin() + mid);   // Nua trai
    vector<int> right(arr.begin() + mid, arr.end());    // Nua phai
    // Buoc 2: Tri (Conquer - Goi de quy)
    // Sap xep de quy nua trai va nua phai
    vector<int> left_sorted = merge_sort(left);
    vector<int> right_sorted = merge_sort(right);
    // Buoc 3: Ket hop (Combine)
    // Tron hai nua da sap xep lai
    return merge(left_sorted, right_sorted);
}
// ----------------- CHAY THU VA MINH HOA -----------------
int main()
{
    vector<int> A = {12, 11, 13, 5, 6, 7};
    cout << "Mang goc: ";
    for (int x : A) cout << x << " ";
    cout << endl;
```

```
    vector<int> sorted_A = merge_sort(A);
    cout << "Mang da sap xep: ";
    for (int x : sorted_A) cout << x << " ";
    cout << endl;
    // Minh hoa qua trinh Merge Sort:
    // [12, 11, 13, 5, 6, 7]
    // Chia thanh: [12, 11, 13] va [5, 6, 7]
    // ... (De quy tiep tuc chia)
    // Merge: [11, 12, 13] va [5, 6, 7]
    // Merge cuoi cung: [5, 6, 7, 11, 12, 13]
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
Mang goc: 12 11 13 5 6 7
Mang da sap xep: 5 6 7 11 12 13
```

- Giải thích Code C++:

```
#include <iostream>             // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>        // Thư viện tổng hợp
using namespace std;            // Tiền tố std
// Ham ket hop (Combine) - Tron hai mang con da sap xep
vector<int> merge(const vector<int>& left, const vector<int>& right)
{
    // Buoc 3: Ham Ket hop (Combine) - Tron hai mang con da sap xep (left va right)
    // thanh mot mang lon da sap xep duy nhat.
    // Do phuc tap thoi gian: O(N)
    vector<int> merged;
    int i = 0, j = 0;  // i la con tro cho mang left, j la con tro cho mang right
    // So sanh va them phan tu nho hon vao merged
    while (i < left.size() && j < right.size())
    {
        if (left[i] <= right[j])
        {
            merged.push_back(left[i]);
            i++;
        }
        else
        {
            merged.push_back(right[j]);
            j++;
        }
    }
    // Them cac phan tu con lai (neu mot trong hai mang da het truoc)
    while (i < left.size())
    {
        merged.push_back(left[i]);
        i++;
    }
    while (j < right.size())
    {
        merged.push_back(right[j]);
        j++;
    }
    return merged;
}
// Ham chinh thuc hien Merge Sort (Chia va Tri)
vector<int> merge_sort(const vector<int>& arr)
{
```

```
    // Ham chinh thuc hien Merge Sort (Chia va Tri)
    // Do phuc tap thoi gian: O(N log N)
    // Dieu kien Dung (Base Case): Mang co 0 hoac 1 phan tu da duoc sap xep
    if (arr.size() <= 1)
        return arr;
    // Buoc 1: Chia (Divide)
    int mid = arr.size() / 2;
    vector<int> left(arr.begin(), arr.begin() + mid);   // Nua trai
    vector<int> right(arr.begin() + mid, arr.end());    // Nua phai
    // Buoc 2: Tri (Conquer - Goi de quy)
    // Sap xep de quy nua trai va nua phai
    vector<int> left_sorted = merge_sort(left);
    vector<int> right_sorted = merge_sort(right);
    // Buoc 3: Ket hop (Combine)
    // Tron hai nua da sap xep lai
    return merge(left_sorted, right_sorted);
}
// ---------------- CHAY THU VA MINH HOA ----------------
int main()
{
    vector<int> A = {12, 11, 13, 5, 6, 7};
    cout << "Mang goc: ";
    for (int x : A) cout << x << " ";
    cout << endl;
    vector<int> sorted_A = merge_sort(A);
    cout << "Mang da sap xep: ";
    for (int x : sorted_A) cout << x << " ";
    cout << endl;
    // Minh hoa qua trinh Merge Sort:
    // [12, 11, 13, 5, 6, 7]
    // Chia thanh: [12, 11, 13] va [5, 6, 7]
    // ... (De quy tiep tuc chia)
    // Merge: [11, 12, 13] va [5, 6, 7]
    // Merge cuoi cung: [5, 6, 7, 11, 12, 13]
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
Mang goc: 12 11 13 5 6 7
Mang da sap xep: 5 6 7 11 12 13
```

VI. Tổng kết & Bài tập Về nhà (10 phút)

1. Tóm tắt Merge Sort:

- **Khái niệm:** Thuật toán sắp xếp dựa trên Chia để trị.
- **Ưu điểm:** Độ phức tạp thời gian luôn là O(N log N) (ổn định và hiệu quả, không bị ảnh hưởng bởi thứ tự ban đầu của dữ liệu).
- **Nhược điểm:** Tốn O(N) không gian bộ nhớ phụ do cần mảng tạm để trộn.

2. Bài tập Về nhà:

2.1. Bài tập 1 (Sắp xếp bằng Merge Sort): Cài đặt Merge Sort để sắp xếp một mảng các đối tượng (ví dụ: danh sách sinh viên) dựa trên một thuộc tính cụ thể (ví dụ: điểm số).

2.1.1. Code Python:

```
# Mẫu dữ liệu mô phỏng đối tượng sinh viên
# Mỗi đối tượng là một dictionary với các thuộc tính 'name' và 'score'
STUDENTS = [
    {'name': 'An', 'score': 85, 'id': 'A101'},
```

```
    {'name': 'Binh', 'score': 92, 'id': 'B202'},
    {'name': 'Cuong', 'score': 78, 'id': 'C303'},
    {'name': 'Dung', 'score': 92, 'id': 'D404'},
    {'name': 'Ede', 'score': 85, 'id': 'E505'}
]
def merge(left, right, key):
    # Hàm trộn hai danh sách đối tượng đã sắp xếp, dựa trên thuộc tính 'key'.
    merged = []
    i = j = 0
    # So sánh và thêm đối tượng có giá trị 'key' nhỏ hơn vào merged
    while i < len(left) and j < len(right):
        # SỰ THAY ĐỔI: So sánh dựa trên giá trị của thuộc tính 'key'
        if left[i][key] <= right[j][key]:
            merged.append(left[i])
            i += 1
        else:
            merged.append(right[j])
            j += 1
    # Thêm các đối tượng còn lại
    merged.extend(left[i:])
    merged.extend(right[j:])
    return merged
def merge_sort_objects(arr, key='score'):
    # Hàm chính thực hiện Merge Sort cho danh sách đối tượng.
    if len(arr) <= 1:
        return arr
    # Bước 1: Chia (Divide)
    mid = len(arr) // 2
    left = arr[:mid]
    right = arr[mid:]
    # Bước 2: Trị (Conquer - gọi đệ quy, truyền key)
    # LƯU Ý: Phải truyền khóa sắp xếp 'key' vào các lời gọi đệ quy
    left_sorted = merge_sort_objects(left, key)
    right_sorted = merge_sort_objects(right, key)
    # Bước 3: Kết hợp (Combine - truyền key)
    return merge(left_sorted, right_sorted, key)
# ----------------- CHẠY THỬ VÀ MINH HỌA -----------------
print("--- MANG GOC ---")
for student in STUDENTS:
    print(student)
# 1. Sắp xếp theo 'score' (Mặc định)
sorted_by_score = merge_sort_objects(STUDENTS.copy(), key='score')
print("\n--- KET QUA 1: Sap xep theo Diem so (Score) ---")
for student in sorted_by_score:
    print(f"Diem: {student['score']} | Ten: {student['name']}")
# 2. Sắp xếp theo 'name'
sorted_by_name = merge_sort_objects(STUDENTS.copy(), key='name')
print("\n--- KET QUA 2: Sap xep theo Ten (Name) ---")
for student in sorted_by_name:
    print(f"Ten: {student['name']} | Diem: {student['score']}")
```

=> Kết quả sau khi chạy chương trình:

```
--- MANG GOC ---
{'name': 'An', 'score': 85, 'id': 'A101'}
{'name': 'Binh', 'score': 92, 'id': 'B202'}
{'name': 'Cuong', 'score': 78, 'id': 'C303'}
{'name': 'Dung', 'score': 92, 'id': 'D404'}
{'name': 'Ede', 'score': 85, 'id': 'E505'}
```

```
--- KET QUA 1: Sap xep theo Diem so (Score)
Diem: 78 | Ten: Cuong
Diem: 85 | Ten: An
Diem: 85 | Ten: Ede
Diem: 92 | Ten: Binh
Diem: 92 | Ten: Dung

--- KET QUA 2: Sap xep theo Ten (Name) ---
Ten: An | Diem: 85
Ten: Binh | Diem: 92
Ten: Cuong | Diem: 78
Ten: Dung | Diem: 92
Ten: Ede | Diem: 85
```

2.1.2. Code C++:

```cpp
#include <iostream>             // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>        // Thư viện tổng hợp
using namespace std;            // Tiền tố std
// Cau truc du lieu mo phong doi tuong sinh vien
struct Student
{
    string name;
    int score;
    string id;
};
// Ham tron hai danh sach doi tuong da sap xep, dua tren thuoc tinh 'key'
vector<Student> merge(const vector<Student>& left,
                      const vector<Student>& right,
                      const string& key)
{
    vector<Student> merged;
    int i = 0, j = 0;
    // So sanh va them doi tuong co gia tri 'key' nho hon vao merged
    while (i < left.size() && j < right.size())
    {
        // SU THAY DOI: So sanh dua tren gia tri cua thuoc tinh 'key'
        bool condition;
        if (key == "score")
            condition = left[i].score <= right[j].score;
        else  // key == "name"
            condition = left[i].name <= right[j].name;
        if (condition)
        {
            merged.push_back(left[i]);
            i++;
        }
        else
        {
            merged.push_back(right[j]);
            j++;
        }
    }
    // Them cac doi tuong con lai
    while (i < left.size())
    {
        merged.push_back(left[i]);
        i++;
```

```
    }
    while (j < right.size())
    {
        merged.push_back(right[j]);
        j++;
    }
    return merged;
}
// Ham chinh thuc hien Merge Sort cho danh sach doi tuong
vector<Student> merge_sort_objects(const vector<Student>& arr,
                                   const string& key = "score")
{
    // Ham chinh thuc hien Merge Sort cho danh sach doi tuong.
    if (arr.size() <= 1)
        return arr;
    // Buoc 1: Chia (Divide)
    int mid = arr.size() / 2;
    vector<Student> left(arr.begin(), arr.begin() + mid);
    vector<Student> right(arr.begin() + mid, arr.end());
    // Buoc 2: Tri (Conquer - goi de quy, truyen key)
    // LUU Y: Phai truyen khoa sap xep 'key' vao cac loi goi de quy
    vector<Student> left_sorted = merge_sort_objects(left, key);
    vector<Student> right_sorted = merge_sort_objects(right, key);
    // Buoc 3: Ket hop (Combine - truyen key)
    return merge(left_sorted, right_sorted, key);
}
// ----------------- CHAY THU VA MINH HOA -----------------
int main()
{
    // Mau du lieu mo phong doi tuong sinh vien
    vector<Student> STUDENTS = {
        {"An", 85, "A101"},
        {"Binh", 92, "B202"},
        {"Cuong", 78, "C303"},
        {"Dung", 92, "D404"},
        {"Ede", 85, "E505"}
    };
    cout << "--- MANG GOC ---" << endl;
    for (const Student& student : STUDENTS)
    {
        cout << "Ten: " << student.name
             << ", Diem: " << student.score
             << ", ID: " << student.id << endl;
    }
    // 1. Sap xep theo 'score' (Mac dinh)
    vector<Student> sorted_by_score =
        merge_sort_objects(STUDENTS, "score");
    cout << "\n--- KET QUA 1: Sap xep theo Diem so (Score) ---" << endl;
    for (const Student& student : sorted_by_score)
    {
        cout << "Diem: " << student.score
             << " | Ten: " << student.name << endl;
    }
    // 2. Sap xep theo 'name'
    vector<Student> sorted_by_name =
        merge_sort_objects(STUDENTS, "name");
    cout << "\n--- KET QUA 2: Sap xep theo Ten (Name) ---" << endl;
    for (const Student& student : sorted_by_name)
```

```
    {
        cout << "Ten: " << student.name
             << " | Diem: " << student.score << endl;
    }
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
--- MANG GOC ---
Ten: An, Diem: 85, ID: A101
Ten: Binh, Diem: 92, ID: B202
Ten: Cuong, Diem: 78, ID: C303
Ten: Dung, Diem: 92, ID: D404
Ten: Ede, Diem: 85, ID: E505

--- KET QUA 1: Sap xep theo Diem so (Score) ---
Diem: 78 | Ten: Cuong
Diem: 85 | Ten: An
Diem: 85 | Ten: Ede
Diem: 92 | Ten: Binh
Diem: 92 | Ten: Dung

--- KET QUA 2: Sap xep theo Ten (Name) ---
Ten: An | Diem: 85
Ten: Binh | Diem: 92
Ten: Cuong | Diem: 78
Ten: Dung | Diem: 92
Ten: Ede | Diem: 85
```

2.2. Bài tập 2 (Đếm cặp nghịch thế): Sử dụng ý tưởng của hàm Merge để giải quyết bài toán đếm số lượng cặp nghịch thế trong một mảng A (A[i] > A[j] với i < j) với độ phức tạp O(N log N).

2.2.1. Code Python:

```python
def merge_and_count_FIXED(arr, L, mid, R):
    # Hàm Trộn và Đếm (Merge and Count) - Đã sửa lỗi lát cắt/chỉ mục.
    # L: chỉ mục bắt đầu
    # mid: chỉ mục cuối của nửa trái
    # R: chỉ mục cuối của mảng gốc
    # Tạo các mảng tạm L và R.
    # Python slicing: [a:b] lấy từ a đến b-1.
    left_half = arr[L : mid + 1]
    right_half = arr[mid + 1 : R + 1]
    i = 0  # Con trỏ cho nửa trái (left_half)
    j = 0  # Con trỏ cho nửa phải (right_half)
    k = L  # Con trỏ chèn vào mảng gốc arr (Bắt đầu từ L)
    inversions = 0
    while i < len(left_half) and j < len(right_half):
        if left_half[i] <= right_half[j]:
            # Không có nghịch thế: Chèn phần tử nhỏ hơn từ nửa trái
            arr[k] = left_half[i]
            i += 1
        else:
            # CÓ nghịch thế: left_half[i] > right_half[j]
            arr[k] = right_half[j]
            j += 1
            # Tính số nghịch thế: Bằng số phần tử còn lại trong nửa trái
```

```
            inversions += (len(left_half) - i)
        k += 1
    # Thêm các phần tử còn lại (nửa trái)
    while i < len(left_half):
        arr[k] = left_half[i]
        i += 1
        k += 1
    # Thêm các phần tử còn lại (nửa phải)
    while j < len(right_half):
        arr[k] = right_half[j]
        j += 1
        k += 1
    return inversions
def count_inversions_FIXED(arr, L, R):
    # Hàm đệ quy chính (Chia và Trị) - Đã sửa lỗi truyền tên hàm
    inversions = 0
    if L < R:
        # 1. Chia (Divide)
        mid = (L + R) // 2
        # 2. Trị (Conquer - đệ quy)
        # Đếm nghịch thế trong nửa trái
        inversions += count_inversions_FIXED(arr, L, mid)
        # Đếm nghịch thế trong nửa phải
        inversions += count_inversions_FIXED(arr, mid + 1, R)
        # 3. Kết hợp (Combine)
        # Đếm nghịch thế GIỮA hai nửa và trộn chúng
        inversions += merge_and_count_FIXED(arr, L, mid, R)
    return inversions
def count_inversions_merge_sort_FIXED(A):
    # Hàm giao diện để gọi thuật toán đếm nghịch thế.
    # Mảng phải được copy để tránh ảnh hưởng đến các lần chạy thử khác,
    # và để hàm đệ quy có thể sắp xếp nó tại chỗ.
    arr_copy = A[:]
    N = len(arr_copy)
    if N == 0:
        return 0
    return count_inversions_FIXED(arr_copy, 0, N - 1)
# ----------------- CHẠY THỬ VỚI CODE ĐÃ SỬA -----------------
A1 = [1, 20, 6, 4, 5]
# Cặp nghịch thế: (20, 6), (20, 4), (20, 5), (6, 4), (6, 5) => Tổng 5
count1 = count_inversions_merge_sort_FIXED(A1)
print(f"Mang A1: {A1}")
print(f"So cap nghich the trong A1: {count1}")
A2 = [1, 2, 3, 4]
count2 = count_inversions_merge_sort_FIXED(A2)
print(f"\nMang A2: {A2}")
print(f"So cap nghich the trong A2: {count2}") # Kết quả: 0
A3 = [4, 3, 2, 1]
count3 = count_inversions_merge_sort_FIXED(A3)
print(f"\nMang A3: {A3}")
print(f"So cap nghich the trong A3: {count3}") # Kết quả: 6
```

=> Kết quả sau khi chạy chương trình:

```
Mang A1: [1, 20, 6, 4, 5]
So cap nghich the trong A1: 5

Mang A2: [1, 2, 3, 4]
So cap nghich the trong A2: 0

Mang A3: [4, 3, 2, 1]
So cap nghich the trong A3: 6
```

2.2.2. Code C++:

```
#include <iostream>             // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>        // Thư viện tổng hợp
using namespace std;            // Tiền tố std
long long merge_and_count_FIXED(vector<int>& arr, int L, int mid, int R)
{
    // Ham Tron va Dem (Merge and Count) - Da sua loi chi muc
    // L: chi muc bat dau
    // mid: chi muc cuoi cua nua trai
    // R: chi muc cuoi cua mang goc
    // Tao cac mang tam trai va phai
    vector<int> left_half(arr.begin() + L, arr.begin() + mid + 1);
    vector<int> right_half(arr.begin() + mid + 1, arr.begin() + R + 1);
    int i = 0;      // Con tro cho nua trai
    int j = 0;      // Con tro cho nua phai
    int k = L;      // Con tro chen vao mang goc
    long long inversions = 0;
    while (i < left_half.size() && j < right_half.size())
    {
        if (left_half[i] <= right_half[j])
        {
            // Khong co nghich the
            arr[k] = left_half[i];
            i++;
        }
        else
        {
            // CO nghich the
            arr[k] = right_half[j];
            j++;
            // So nghich the bang so phan tu con lai trong nua trai
            inversions += (left_half.size() - i);
        }
        k++;
    }
    // Them cac phan tu con lai cua nua trai
    while (i < left_half.size())
    {
        arr[k] = left_half[i];
        i++;
        k++;
    }
    // Them cac phan tu con lai cua nua phai
    while (j < right_half.size())
    {
        arr[k] = right_half[j];
        j++;
        k++;
    }
```

```
    return inversions;
}
long long count_inversions_FIXED(vector<int>& arr, int L, int R)
{
    // Ham de quy chinh (Chia va Tri)
    long long inversions = 0;
    if (L < R)
    {
        // 1. Chia (Divide)
        int mid = (L + R) / 2;
        // 2. Tri (Conquer)
        inversions += count_inversions_FIXED(arr, L, mid);
        inversions += count_inversions_FIXED(arr, mid + 1, R);
        // 3. Ket hop (Combine)
        inversions += merge_and_count_FIXED(arr, L, mid, R);
    }
    return inversions;
}
long long count_inversions_merge_sort_FIXED(const vector<int>& A)
{
    // Ham giao dien de goi thuat toan dem nghich the
    // Copy mang de tranh anh huong den du lieu goc
    vector<int> arr_copy = A;
    int N = arr_copy.size();
    if (N == 0)
        return 0;
    return count_inversions_FIXED(arr_copy, 0, N - 1);
}
// ----------------- CHAY THU VOI CODE DA SUA -----------------
int main()
{
    vector<int> A1 = {1, 20, 6, 4, 5};
    // Cac cap nghich the:
    // (20,6), (20,4), (20,5), (6,4), (6,5) => 5
    long long count1 = count_inversions_merge_sort_FIXED(A1);
    cout << "Mang A1: ";
    for (int x : A1) cout << x << " ";
    cout << "\nSo cap nghich the trong A1: " << count1 << endl;
    vector<int> A2 = {1, 2, 3, 4};
    long long count2 = count_inversions_merge_sort_FIXED(A2);
    cout << "\nMang A2: ";
    for (int x : A2) cout << x << " ";
    cout << "\nSo cap nghich the trong A2: " << count2 << endl;
    vector<int> A3 = {4, 3, 2, 1};
    long long count3 = count_inversions_merge_sort_FIXED(A3);
    cout << "\nMang A3: ";
    for (int x : A3) cout << x << " ";
    cout << "\nSo cap nghich the trong A3: " << count3 << endl;
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
Mang A1: 1 20 6 4 5
So cap nghich the trong A1: 5

Mang A2: 1 2 3 4
So cap nghich the trong A2: 0

Mang A3: 4 3 2 1
So cap nghich the trong A3: 6
```

BÀI 33. THAM LAM & CHIA ĐỂ TRỊ: BÀI TOÁN TỐI ƯU CƠ BẢN

I. Mục Tiêu:

- **Kiến thức:** HS phân biệt được rõ ràng khi nào nên sử dụng Tham lam (Greedy) và khi nào nên sử dụng Chia để trị (Divide and Conquer). Nắm được các điều kiện cốt lõi để áp dụng mỗi phương pháp.
- **Kỹ năng:** HS cài đặt được cả hai phương pháp để giải quyết các vấn đề tối ưu hóa khác nhau (tìm kiếm, sắp xếp, lựa chọn).
- **Tư duy:** HS rèn luyện khả năng phân tích bài toán để xác định xem cấu trúc tối ưu của nó phù hợp với chiến lược Tham lam (tối ưu cục bộ) hay Chia để trị (tổng hợp lời giải con).

II. Phân Bổ Thời Gian (90 Phút):

Hoạt Động	Nội Dung	Thời gian (Phút)
Ôn tập & So sánh	So sánh Tham lam và Chia để trị (Nguyên tắc, Độ phức tạp)	15
Lý thuyết 1	Bài toán 1. Tìm kiếm (Binary Search - Chia để trị)	25
Thực hành 1	Cài đặt Binary Search (Python & C++)	20
Lý thuyết 2 & Thực hành 2	Bài toán 2. Lựa chọn (Fractional Knapsack - Tham lam)	30
Tổng kết	Kết luận chung về các phương pháp	5
Tổng cộng		**90**

III. Lý Thuyết 1: So sánh Tham lam và Chia để trị (15 phút)

Tiêu chí	Thuật toán Tham lam (Greedy)	Thuật toán Chia để trị (D&C)
Nguyên tắc cốt lõi	Lựa chọn tối ưu cục bộ tại mỗi bước với hy vọng đạt tối ưu toàn cục.	Phân tách bài toán lớn thành các bài toán con, giải quyết các bài toán con và kết hợp lời giải.
Cấu trúc	Duyệt vòng lặp/Đệ quy đơn giản. Ra quyết định *chỉ một lần* và không quay lại.	Thường là Đệ quy theo kiểu cây (tree recursion).

Điều kiện áp dụng	Phải thỏa mãn **Greedy Choice Property** và **Optimal Substructure**.	Phải thỏa mãn **Optimal Substructure** và bài toán con phải độc lập.
Ví dụ điển hình	Chọn việc, Đổi tiền (Hệ thống tiêu chuẩn), Knapsack phân đoạn.	Merge Sort, Quick Sort, Binary Search, Nhân ma trận Strassen.

IV. Lý thuyết & Thực hành 1: Binary Search (Chia để trị) (45 phút)

1. Bài toán và Nguyên lý Chia để trị:

- **Bài toán:** Tìm kiếm vị trí của một phần tử **x** trong một mảng đã được sắp xếp **A**.
- **Chiến lược Chia để trị:**
  + **Chia (Divide):** So sánh **x** với phần tử ở giữa (**A[mid]**).
  + **Trị (Conquer):** Dựa trên kết quả so sánh, ta loại bỏ một nửa mảng (trái hoặc phải) và chỉ gọi đệ quy trên nửa còn lại (bài toán con).
  + **Kết hợp (Combine):** Không cần bước này, vì lời giải (vị trí) được trả về trực tiếp từ bài toán con.
- **Độ phức tạp:** T(N) = T(N/2) + O(1) => O(log N).

2. Code tham khảo Binary Search:

2.1. Code Python:

```python
def binary_search(arr, target, low, high):
    # Thực hiện Tìm kiếm Nhị phân bằng đệ quy.
    # Tham số:
    # arr (list): Mảng đã được sắp xếp.
    # target (int): Giá trị cần tìm.
    # low (int): Chỉ mục bắt đầu của phạm vi tìm kiếm.
    # high (int): Chỉ mục kết thúc của phạm vi tìm kiếm.
    # Trả về:
    # int: Chỉ mục của target trong mảng, hoặc -1 nếu không tìm thấy.
    # Điều kiện Dừng (Base Case): Phạm vi tìm kiếm không hợp lệ
    if high >= low:
        # Bước 1: Chia (Divide)
        # Tìm chỉ mục giữa của phạm vi hiện tại
        mid = (high + low) // 2
        # Bước 2: Trị (Conquer)
        # Trường hợp 1: Tìm thấy (Lựa chọn tối ưu cục bộ dẫn đến lời giải)
        if arr[mid] == target:
            return mid  # Thành công, trả về chỉ mục
        # Trường hợp 2: Target nằm ở nửa trái
        elif arr[mid] > target:
            # Loại bỏ nửa phải và gọi đệ quy trên phạm vi [low, mid - 1]
            return binary_search(arr, target, low, mid - 1)
        # Trường hợp 3: Target nằm ở nửa phải
        else:
            # Loại bỏ nửa trái và gọi đệ quy trên phạm vi [mid + 1, high]
            return binary_search(arr, target, mid + 1, high)
    else:
        # Thất bại: high < low, nghĩa là phần tử không tồn tại trong mảng
        return -1
# ---------------- CHẠY THỬ VÀ MINH HỌA ----------------
A = [2, 5, 8, 12, 16, 23, 38]
# 1. Trường hợp thành công
target1 = 23
```

```python
idx = binary_search(A, target1, 0, len(A) - 1)
print(f"Mang: {A}")
print(f"Tim kiem {target1}: Vi tri {idx}") # Kết quả: 5
# 2. Trường hợp thất bại
target2 = 10
idx_fail = binary_search(A, target2, 0, len(A) - 1)
print(f"Tim kiem {target2}: Vi tri {idx_fail}") # Kết quả: -1
```

=> Kết quả sau khi chạy chương trình:

```
Mang: [2, 5, 8, 12, 16, 23, 38]
Tim kiem 23: Vi tri 5
Tim kiem 10: Vi tri -1
```

- Giải thích Code Python:

+ **Điều kiện bắt buộc:** Khi liên quan đến tìm kiếm nhị phân thì mảng phải được sắp xếp trước.

+ **Nguyên lý D&C:** Thuật toán luôn loại bỏ một nửa dữ liệu tại mỗi bước so sánh, chia không gian tìm kiếm thành các bài toán con độc lập. **O(log N)**

2.2. Code C++:

```cpp
#include <iostream>             // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>        // Thư viện tổng hợp
using namespace std;            // Tiền tố std
int binary_search(const vector<int>& arr, int target, int low, int high)
{
    // Thuc hien Tim kiem Nhi phan bang de quy
    // Tham so:
    // arr (vector): Mang da duoc sap xep
    // target (int): Gia tri can tim
    // low (int): Chi muc bat dau cua pham vi tim kiem
    // high (int): Chi muc ket thuc cua pham vi tim kiem
    // Tra ve:
    // int: Chi muc cua target trong mang, hoac -1 neu khong tim thay
    // Dieu kien Dung (Base Case): Pham vi tim kiem hop le
    if (high >= low)
    {
        // Buoc 1: Chia (Divide)
        // Tim chi muc giua cua pham vi hien tai
        int mid = (high + low) / 2;
        // Buoc 2: Tri (Conquer)
        // Truong hop 1: Tim thay
        if (arr[mid] == target)
        {
            return mid; // Thanh cong
        }
        // Truong hop 2: Target nam o nua trai
        else if (arr[mid] > target)
        {
            // Goi de quy tren nua trai [low, mid - 1]
            return binary_search(arr, target, low, mid - 1);
        }
        // Truong hop 3: Target nam o nua phai
        else
        {
            // Goi de quy tren nua phai [mid + 1, high]
            return binary_search(arr, target, mid + 1, high);
        }
    }
```

```
    else
    {
        // That bai: Phan tu khong ton tai trong mang
        return -1;
    }
}
// ----------------- CHAY THU VA MINH HOA -----------------
int main()
{
    vector<int> A = {2, 5, 8, 12, 16, 23, 38};
    // 1. Truong hop thanh cong
    int target1 = 23;
    int idx = binary_search(A, target1, 0, A.size() - 1);
    cout << "Mang: ";
    for (int x : A) cout << x << " ";
    cout << endl;
    cout << "Tim kiem " << target1 << ": Vi tri " << idx << endl;
    // 2. Truong hop that bai
    int target2 = 10;
    int idx_fail = binary_search(A, target2, 0, A.size() - 1);
    cout << "Tim kiem " << target2 << ": Vi tri " << idx_fail << endl;
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
Mang: 2 5 8 12 16 23 38
Tim kiem 23: Vi tri 5
Tim kiem 10: Vi tri -1
```

- Giải thích Code C++:

+ Điều kiện bắt buộc: Khi liên quan đến tìm kiếm nhị phân thì mảng phải được sắp xếp trước.

+ Tránh tràn số: C++ thường tính **mid** là **low + (high - low) / 2** để đảm bảo không bị tràn số khi **low** và **high** là các giá trị lớn. Cấu trúc đệ quy thể hiện rõ tính chất chia đôi bài toán.

V. Lý thuyết & Thực hành 2: Fractional Knapsack (Tham lam) (30 phút)

1. Bài toán và Nguyên lý Tham lam:

- Bài toán (Knapsack Phân đoạn): Cho một cái túi có sức chứa **W** và **N** vật phẩm, mỗi vật phẩm có trọng lượng $\mathbf{w_i}$ và giá trị $\mathbf{v_i}$. Ta có thể lấy **một phần** của vật phẩm. Mục tiêu là tối đa hóa tổng giá trị.

- Chiến lược Tham lam Tối ưu: Luôn chọn vật phẩm có **giá trị/trọng lượng (v_i/w_i) lớn nhất** trước:

+ Tính tỷ lệ giá trị/trọng lượng cho mọi vật phẩm.

+ Sắp xếp vật phẩm theo tỷ lệ này giảm dần.

+ Lần lượt chọn các vật phẩm: Lấy toàn bộ vật phẩm nếu còn đủ chỗ; nếu không, lấy phần còn lại để lấp đầy túi.

- Độ phức tạp: O(N log N) (chủ yếu do bước sắp xếp).

2. Code tham khảo Fractional Knapsack:

2.1. Code Python:

```
class Item:
    # Định nghĩa một Vật phẩm với giá trị (value), trọng lượng (weight),
    # và tỷ lệ giá trị/trọng lượng (ratio).
    def __init__(self, value, weight):
```

```
        self.value = value
        self.weight = weight
        # Tính toán tỷ lệ giá trị/trọng lượng (Chiến lược Tham lam)
        self.ratio = value / weight
    def __repr__(self):
        # Hàm giúp in đối tượng Item dễ đọc hơn
        return f"Item(V={self.value}, W={self.weight}, Ratio={self.ratio:.2f})"
def fractional_knapsack(W, items):
    # Giải bài toán Fractional Knapsack bằng thuật toán Tham lam.
    # Tham số:
    # W (float/int): Sức chứa tối đa của cái túi (Knapsack capacity).
    # items (list): Danh sách các đối tượng Item.
    # Trả về:
    # float: Tổng giá trị tối đa có thể đạt được.
    # 1. SẮP XẾP THAM LAM: Theo tỷ lệ value/weight (ratio) giảm dần (O(N log N))
    # Chiến lược: Luôn chọn vật phẩm có tỷ lệ giá trị/trọng lượng cao nhất trước.
    items.sort(key=lambda x: x.ratio, reverse=True)
    total_value = 0.0
    remaining_W = W  # Trọng lượng còn lại của túi
    print(f"--- Qua trinh Tham lam (W con lai: {W}) ---")
    for item in items:
        if remaining_W == 0:
            print("Tui da day, dung lai.")
            break
        # 2. LỰA CHỌN THAM LAM
        if item.weight <= remaining_W:
            # Trường hợp 1: Lấy TOÀN BỘ vật phẩm (vì còn đủ chỗ)
            total_value += item.value
            remaining_W -= item.weight
            print(f"-> Chon toan bo {item.weight}kg (Ratio {item.ratio:.2f}). W
con: {remaining_W}")
        else:
            # Trường hợp 2: Lấy MỘT PHẦN vật phẩm (Fractional)
            # Tính toán phần trăm (phân đoạn) có thể lấy
            fraction = remaining_W / item.weight
            # Tính toán giá trị và trọng lượng lấy được
            value_taken = item.value * fraction
            weight_taken = item.weight * fraction
            total_value += value_taken
            remaining_W = 0 # Túi đầy sau khi lấy phần này
            print(f"-> Chi lay {weight_taken:.2f}kg ({fraction*100:.1f}%) cua vat
pham (Ratio {item.ratio:.2f}). Tui DAY.")
    return total_value
# ----------------- CHẠY THỬ VÀ MINH HỌA -----------------
# Dữ liệu vật phẩm: (Giá trị, Trọng lượng)
items_data = [
    (60, 10),  # Ratio: 60/10 = 6.0
    (100, 20), # Ratio: 100/20 = 5.0
    (120, 30)  # Ratio: 120/30 = 4.0
]
W = 50
# Khởi tạo các đối tượng Item
items = [Item(v, w) for v, w in items_data]
# Chạy thuật toán
max_val = fractional_knapsack(W, items)
print(f"\n==========================================")
print(f"Suc chua tui (W): {W}")
print(f"Tong gia tri toi da dat duoc: {max_val:.2f}")
```

```
# Chi tiết lời giải tối ưu:
# 1. Item 1 (Ratio 6.0): Lấy 10kg -> Value: 60. W còn: 40.
# 2. Item 2 (Ratio 5.0): Lấy 20kg -> Value: 100. W còn: 20.
# 3. Item 3 (Ratio 4.0): Chỉ lấy 20kg (còn lại) / 30kg * 120 = 80. W còn: 0.
# Tổng giá trị: 60 + 100 + 80 = 240.0
```

=> Kết quả sau khi chạy chương trình:

```
--- Qua trinh Tham lam (W con lai: 50) ---
-> Chon toan bo 10kg (Ratio 6.00). W con: 40
-> Chon toan bo 20kg (Ratio 5.00). W con: 20
-> Chi lay 20.00kg (66.7%) cua vat pham (Ratio 4.00). Tui DAY.

=============================================
Suc chua tui (W): 50
Tong gia tri toi da dat duoc: 240.00
```

- Giải thích Code Python:

Tối ưu hóa Tham lam: Sắp xếp là bước quyết định. Nếu không sắp xếp theo tỷ lệ, thuật toán Tham lam sẽ thất bại. Việc lặp qua danh sách đã sắp xếp thể hiện nguyên tắc Tham lam: luôn chọn tốt nhất có thể tại thời điểm hiện tại.

2.2. Code C++:

```
#include <iostream>             // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>        // Thư viện tổng hợp
using namespace std;            // Tiền tố std
class Item
{
public:
    // Dinh nghia mot Vat pham voi gia tri (value), trong luong (weight),
    // va ty le gia tri/trong luong (ratio)
    double value;
    double weight;
    double ratio;
    Item(double v, double w)
    {
        value = v;
        weight = w;
        // Tinh toan ty le gia tri/trong luong (Chien luoc Tham lam)
        ratio = value / weight;
    }
    // Ham giup in doi tuong Item de doc hon
    void print() const
    {
        cout << "Item(V=" << value
             << ", W=" << weight
             << ", Ratio=" << fixed << setprecision(2) << ratio << ")";
    }
};
double fractional_knapsack(double W, vector<Item>& items)
{
    // Giai bai toan Fractional Knapsack bang thuat toan Tham lam
    // Tham so:
    // W (double): Suc chua toi da cua tui
    // items (vector<Item>): Danh sach cac vat pham
    // Tra ve:
    // double: Tong gia tri toi da co the dat duoc
    // 1. SAP XEP THAM LAM: Theo ty le value/weight giam dan (O(N log N))
    sort(items.begin(), items.end(),
         [](const Item& a, const Item& b)
```

```
        {
            return a.ratio > b.ratio;
        });
    double total_value = 0.0;
    double remaining_W = W;
    cout << "--- Qua trinh Tham lam (W con lai: " << W << ") ---" << endl;
    for (const Item& item : items)
    {
        if (remaining_W == 0)
        {
            cout << "Tui da day, dung lai." << endl;
            break;
        }
        // 2. LUA CHON THAM LAM
        if (item.weight <= remaining_W)
        {
            // Truong hop 1: Lay TOAN BO vat pham
            total_value += item.value;
            remaining_W -= item.weight;
            cout << "-> Chon toan bo "
                 << item.weight << "kg (Ratio "
                 << fixed << setprecision(2) << item.ratio
                 << "). W con: " << remaining_W << endl;
        }
        else
        {
            // Truong hop 2: Lay MOT PHAN vat pham (Fractional)
            double fraction = remaining_W / item.weight;
            double value_taken = item.value * fraction;
            double weight_taken = item.weight * fraction;
            total_value += value_taken;
            remaining_W = 0;
            cout << "-> Chi lay "
                 << fixed << setprecision(2) << weight_taken
                 << "kg (" << fraction * 100
                 << "%) cua vat pham (Ratio "
                 << item.ratio << "). Tui DAY." << endl;
        }
    }
    return total_value;
}
// ----------------- CHAY THU VA MINH HOA -----------------
int main()
{
    // Du lieu vat pham: (Gia tri, Trong luong)
    vector<pair<double, double>> items_data = {
        {60, 10},   // Ratio: 6.0
        {100, 20},  // Ratio: 5.0
        {120, 30}   // Ratio: 4.0
    };
    double W = 50;
    // Khoi tao cac doi tuong Item
    vector<Item> items;
    for (auto& p : items_data)
    {
        items.push_back(Item(p.first, p.second));
    }
    // Chay thuat toan
```

```
    double max_val = fractional_knapsack(W, items);
    cout << "\n=========================================" << endl;
    cout << "Suc chua tui (W): " << W << endl;
    cout << "Tong gia tri toi da dat duoc: "
         << fixed << setprecision(2) << max_val << endl;
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
--- Qua trinh Tham lam (W con lai: 50) ---
-> Chon toan bo 10kg (Ratio 6.00). W con: 40.00
-> Chon toan bo 20.00kg (Ratio 5.00). W con: 20.00
-> Chi lay 20.00kg (66.67%) cua vat pham (Ratio 4.00). Tui DAY.

=========================================
Suc chua tui (W): 50.00
Tong gia tri toi da dat duoc: 240.00
```

- Giải thích Code C++:

Do sử dụng **double** cho tỷ lệ và tổng giá trị, ta cần cẩn thận khi ép kiểu (casting) trong tính toán.

VI. Tổng kết & Bài tập Về nhà (5 phút):

1. Tổng kết:

- **Tham lam:** Tối ưu khi lựa chọn cục bộ luôn dẫn đến tối ưu toàn cục. Thường dùng cho các bài toán lựa chọn (Selection/Optimization).

- **Chia để trị:** Tối ưu khi bài toán có thể chia thành các bài toán con độc lập. Thường dùng cho các bài toán Sắp xếp, Tìm kiếm, Xử lý dữ liệu quy mô lớn.

2. Bài tập Về nhà: So sánh Fractional Knapsack (Tham lam) với 0/1 Knapsack (Quy hoạch động) và giải thích tại sao 0/1 Knapsack không thể dùng Tham lam.

- Đây là phần so sánh chi tiết giữa bài toán **Fractional Knapsack (Tham lam)** và **0/1 Knapsack (Quy hoạch động)**, cùng với giải thích lý do tại sao thuật toán Tham lam thất bại trong bài toán 0/1 Knapsack.

- Ta sẽ minh họa sự thất bại của chiến lược Tham lam trong bài toán 0/1 Knapsack.

2.1. Phân biệt Hai Bài toán Knapsack:

Tiêu chí	Fractional Knapsack (Phân đoạn)	0/1 Knapsack (Không phân đoạn)
Bản chất	Có thể lấy **một phần** của vật phẩm.	Chỉ có thể lấy **toàn bộ** (1) hoặc **không lấy** (0) vật phẩm.
Phương pháp giải	**Tham lam (Greedy Algorithm)**	**Quy hoạch động (Dynamic Programming - DP)**
Tính tối ưu	Tham lam **luôn tối ưu**.	Tham lam **thất bại**. DP là phương pháp tối ưu.
Độ phức tạp	O(N log N) (chủ yếu do bước sắp xếp).	O(N . W) (với N là số vật phẩm, W là sức chứa túi).

2.2. Giải thích tại sao Tham lam thất bại trong 0/1 Knapsack:

- Thuật toán Tham lam dựa vào nguyên tắc **Lựa chọn Tham lam (Greedy Choice Property)**, tức là lựa chọn tối ưu cục bộ sẽ dẫn đến tối ưu toàn cục.

- **Lý do thất bại:**

+ **Thiếu Cấu trúc Tối ưu Con Độc lập:** Trong 0/1 Knapsack, quyết định chọn một vật phẩm **(I_k)** có tỷ lệ giá trị/trọng lượng cao nhất có thể **chiếm hết** hoặc gần hết sức chứa còn lại **(W')**, ngăn cản việc chọn các vật phẩm khác có tổng giá trị lớn hơn.

+ **Mất Tối ưu Toàn cục:** Lựa chọn tối ưu cục bộ (chọn vật phẩm I_A tốt nhất) có thể là một quyết định tồi trong dài hạn.

- **Ví dụ minh họa:** Xét túi có sức chứa W = 50.

Vật phẩm	Giá trị (vi)	Trọng lượng (wi)	Tỷ lệ (vi/wi)
A	60	10	6.0 (Cao nhất)
B	100	20	5.0
C	120	30	4.0

- **Chiến lược Tham lam (theo Tỷ lệ giảm dần):**

+ Chọn A (60, 10). W còn: 50 - 10 = 40. **Giá trị: 60.**

+ Chọn B (100, 20). W còn: 40 - 20 = 20. **Giá trị: 60 + 100 = 160.**

+ Chọn C (120, 30). 30 > 20. **Không thể chọn.**

+ **Tổng giá trị Tham lam: 160.**

- **Lời giải Tối ưu (Optimal):**

+ Bỏ qua A.

+ Chọn B (100, 20) và C (120, 30). 20 + 30 = 50 ≤ W.

+ **Tổng giá trị Tối ưu: 100 + 120 = 220.**

- Rõ ràng, chiến lược Tham lam (luôn chọn A) dẫn đến tổng giá trị 160, nhỏ hơn nhiều so với tổng giá trị tối ưu 220.

2.3. Code Minh họa sự thất bại:

2.3.1. Code Python:

```
class Item:
    # Định nghĩa vật phẩm cho bài toán Knapsack.
    def __init__(self, value, weight):
        self.value = value
        self.weight = weight
        self.ratio = value / weight
    def __repr__(self):
        return f"Item(V={self.value}, W={self.weight}, Ratio={self.ratio:.2f})"
def greedy_for_01_knapsack(W, items):
    # Thực hiện chiến lược Tham lam (sắp xếp theo tỷ lệ) cho bài toán 0/1 Knapsack.
    # (Lưu ý: Đây không phải là giải pháp tối ưu cho 0/1 Knapsack).
    # 1. Chiến lược Tham lam: Sắp xếp theo tỷ lệ giá trị/trọng lượng giảm dần
    items.sort(key=lambda x: x.ratio, reverse=True)
    total_value = 0
    remaining_W = W
    selected_items = []
    print(f"--- Ap dung Tham lam (Sap xep theo Ratio) cho 0/1 Knapsack ---")
    for item in items:
        if remaining_W == 0:
            break
        # Quyết định Tham lam: Chỉ lấy toàn bộ (0/1)
```

```
        if item.weight <= remaining_W:
            total_value += item.value
            remaining_W -= item.weight
            selected_items.append(item)
            print(f"-> Chon toan bo {item} (W con: {remaining_W})")
        else:
            # KHÔNG THỂ lấy một phần, nên BỎ QUA vật phẩm này
            print(f"-> Bo qua {item} vi W={item.weight} > W con:
{remaining_W}")
    return total_value, selected_items
# --- CHẠY THỬ VÀ SO SÁNH ---
W = 50
items = [
    Item(60, 10),  # A: Ratio 6.0 (Cao nhất)
    Item(100, 20), # B: Ratio 5.0
    Item(120, 30)  # C: Ratio 4.0 (Thấp nhất)
]
# 1. Chạy Tham lam (Greedy)
greedy_value, greedy_selection = greedy_for_01_knapsack(W, items)
print("\n[Tham Lam]:")
print(f"Tong gia tri Tham lam (Chon A và B): {greedy_value}")
print(f"W con lại: {W - sum(i.weight for i in greedy_selection)}")
# 2. Lời giải Tối ưu (Optimal)
# Tối ưu là chọn B và C: 100 + 120 = 220
optimal_value = 220
optimal_selection = [i for i in items if i.value in [100, 120]]
print("\n[Toi Uu - DP]:")
print(f"Tong gia tri Toi uu (Chon B va C): {optimal_value}")
print(f"W con lai: {W - sum(i.weight for i in optimal_selection)}")
print(f"\n=> Ket luan: Tham lam ({greedy_value}) < Toi uu ({optimal_value}).")
```

=> Kết quả sau khi chạy chương trình:

```
--- Ap dung Tham lam (Sap xep theo Ratio) cho 0/1 Knapsack ---
-> Chon toan bo Item(V=60, W=10, Ratio=6.00) (W con: 40)
-> Chon toan bo Item(V=100, W=20, Ratio=5.00) (W con: 20)
-> Bo qua Item(V=120, W=30, Ratio=4.00) vi W=30 > W con: 20

[Tham Lam]:
Tong gia tri Tham lam (Chon A và B): 160
W con lại: 20

[Toi Uu - DP]:
Tong gia tri Toi uu (Chon B va C): 220
W con lai: 0

=> Ket luan: Tham lam (160) < Toi uu (220).
```

2.3.2. Code C++:

```
#include <iostream>             // Thư viện nhập, xuất dữ liệu
#include <bits/stdc++.h>        // Thư viện tổng hợp
using namespace std;            // Tiền tố std
class Item
{
public:
    // Dinh nghia vat pham cho bai toan Knapsack
    int value;
    int weight;
    double ratio;
    Item(int v, int w)
    {
        value = v;
```

```
        weight = w;
        ratio = (double)value / weight;
    }
    // Tuong duong ham __repr__ trong Python
    void print() const
    {
        cout << "Item(V=" << value
             << ", W=" << weight
             << ", Ratio=" << ratio << ")";
    }
};
pair<int, vector<Item>> greedy_for_01_knapsack(int W, vector<Item> items)
{
    // Thuc hien chien luoc Tham lam (sap xep theo ty le) cho bai toan 0/1 Knapsack
    // (Luu y: Day khong phai la giai phap toi uu cho 0/1 Knapsack)
    // 1. Chien luoc Tham lam: Sap xep theo ty le gia tri/trong luong giam dan
    sort(items.begin(), items.end(),
         [](const Item& a, const Item& b)
         {
             return a.ratio > b.ratio;
         });
    int total_value = 0;
    int remaining_W = W;
    vector<Item> selected_items;
    cout << "--- Ap dung Tham lam (Sap xep theo Ratio) cho 0/1 Knapsack ---" <<
endl;
    for (const Item& item : items)
    {
        if (remaining_W == 0)
            break;
        // Quyet dinh Tham lam: Chi lay toan bo (0/1)
        if (item.weight <= remaining_W)
        {
            total_value += item.value;
            remaining_W -= item.weight;
            selected_items.push_back(item);
            cout << "-> Chon toan bo ";
            item.print();
            cout << " (W con: " << remaining_W << ")" << endl;
        }
        else
        {
            // KHONG THE lay mot phan, nen BO QUA vat pham nay
            cout << "-> Bo qua ";
            item.print();
            cout << " vi W=" << item.weight
                 << " > W con: " << remaining_W << endl;
        }
    }
    return {total_value, selected_items};
}
// --- CHAY THU VA SO SANH ---
int main()
{
    int W = 50;
    vector<Item> items = {
        Item(60, 10),   // A: Ratio 6.0 (Cao nhat)
        Item(100, 20),  // B: Ratio 5.0
```

```
        Item(120, 30)   // C: Ratio 4.0 (Thap nhat)
    };
    // 1. Chay Tham lam (Greedy)
    auto result = greedy_for_01_knapsack(W, items);
    int greedy_value = result.first;
    vector<Item> greedy_selection = result.second;
    cout << "\n[Tham Lam]:" << endl;
    cout << "Tong gia tri Tham lam (Chon A va B): "
         << greedy_value << endl;
    int used_weight = 0;
    for (const Item& i : greedy_selection)
        used_weight += i.weight;
    cout << "W con lai: " << W - used_weight << endl;
    // 2. Loi giai Toi uu (Optimal)
    // Toi uu la chon B va C: 100 + 120 = 220
    int optimal_value = 220;
    vector<Item> optimal_selection;
    for (const Item& i : items)
    {
        if (i.value == 100 || i.value == 120)
            optimal_selection.push_back(i);
    }
    cout << "\n[Toi Uu - DP]:" << endl;
    cout << "Tong gia tri Toi uu (Chon B va C): "
         << optimal_value << endl;
    int optimal_weight = 0;
    for (const Item& i : optimal_selection)
        optimal_weight += i.weight;
    cout << "W con lai: " << W - optimal_weight << endl;
    cout << "\n=> Ket luan: Tham lam ("
         << greedy_value << ") < Toi uu ("
         << optimal_value << ")." << endl;
    return 0;
}
```

=> Kết quả sau khi chạy chương trình:

```
--- Ap dung Tham lam (Sap xep theo Ratio) cho 0/1 Knapsack ---
-> Chon toan bo Item(V=60, W=10, Ratio=6) (W con: 40)
-> Chon toan bo Item(V=100, W=20, Ratio=5) (W con: 20)
-> Bo qua Item(V=120, W=30, Ratio=4) vi W=30 > W con: 20

[Tham Lam]:
Tong gia tri Tham lam (Chon A va B): 160
W con lai: 20

[Toi Uu - DP]:
Tong gia tri Toi uu (Chon B va C): 220
W con lai: 0

=> Ket luan: Tham lam (160) < Toi uu (220).
```

*- Bộ tài liệu này gồm hơn **700 trang**, được biên soạn nhằm phục vụ cho ôn luyện thi học sinh giỏi. Hiện tại, tác giả **xin chia sẻ trước 500 trang** để nhận ý kiến đóng góp và đánh giá từ cộng đồng.*

*- Nếu tài liệu nhận được sự quan tâm và đón nhận, **phần nội dung còn lại sẽ được tiếp tục chia sẻ miễn phí trong thời gian tới.***

- Rất mong nhận được ý kiến đóng góp chân thành của quý thầy cô và các bạn.

PHẦN 2. LUYỆN GIẢI ĐỀ THI THEO CẤU TRÚC TỈNH

ĐỀ SỐ 1
(Học sinh giỏi 9 – Đà Nẵng – Năm học: 2024 – 2025)

ĐỀ THI HỌC SINH GIỎI LỚP 9 MÔN TIN HỌC
THÀNH PHỐ ĐÀ NẴNG
NAM HỌC 2024 – 2025

Thời gian làm bài: ... phút, không kể thời gian phát đề

TỔNG QUAN ĐỀ THI

Bài	Tên bài	Tên file CT	File input	File output	Điểm
1	Kí tự	KITU.*	KITU.INP	TONG2.OUT	100
2	Số tròn chục	SOTC.*	SOTC.INP	SOTC.OUT	100
3	Tổng liên tiếp	TONGLT.*	TONGLT.INP	TONGLT.OUT	100
4	Chiến binh	CHIENBINH.*	CHIENBINH.INP	CHIENBINH.OUT	100

*Dấu * được thay thế bởi PAS, CPP hoặc PY tương ứng với ngôn ngữ lập trình sử dụng*

LẬP TRÌNH GIẢI CÁC BÀI TOÁN SAU

Bài 10. KITU2 Kí tự 2

Cho một xâu s chỉ chứa các ký tự in hoa trong bảng chữ cái tiếng Anh **A...Z.**

Yêu cầu: Hãy kiểm tra xâu s còn thiếu những ký tự nào trong bảng chữ cái tiếng Anh **A...Z.**

Input: Một dòng duy nhất chứa xâu s có độ dài không quá 10^5 ký tự.

Output: In ra các ký tự chưa xuất hiện trong xâu s, các ký tự được viết liền tục theo thứ tự tăng dần.

Ví dụ:

Input	Output	Giải thích
EFGHJABCD	IKLMNOPQRSTUVWXYZ	

Bài 11. SOTC Số tròn chục

Số tròn chục là số có chữ số hàng đơn vị là chữ số **0.**

Cho hai số tự nhiên L và R. Hãy đếm xem có bao nhiêu số tròn chục lớn hơn **L** và nhỏ hơn **R.**

Dữ liệu: Nhập vào số tự nhiên L, R ($L < R \le 10^{12}$). Mỗi số trên một dòng.

Kết quả: Ghi ra kết quả của bài toán.

Ví dụ:

Input	Output	Giải thích
5 31	3	Có 3 số tròn chục lớn hơn 5 và nhỏ hơn 31 là: 10, 20, 30

Chấm điểm:

- Nếu chương trình chạy đúng những trường hợp $R \le 10^6$, thí sinh sẽ được 80 điểm;
- Nếu chương trình chạy đúng những trường hợp $R \le 10^{12}$, thí sinh sẽ được 100 điểm.

Bài 12. TONGLT1 Tổng Liên Tiếp 1

Trong cuộc thi "Học sinh tài năng" được tổ chức tại một trường học, ban tổ chức chuẩn bị một bảng điểm điện tử để hiển thị điểm số của từng thí sinh. Điểm số của **n** thí sinh được hiển thị theo thứ tự từ thí sinh **1** đến thí sinh **n**, sau đó lặp lại vô hạn lần. Cụ thể, sau khi hiển thị điểm của thí sinh **n**, bảng điểm sẽ quay lại hiển thị điểm của thí sinh **1** rồi thí sinh **2**, và cứ thế không ngừng.

Yêu cầu: Hãy giúp ban tổ chức tính tổng **k** điểm số liên tiếp xuất hiện trên bảng điểm, bắt đầu từ vị trí **p**.

Input

- Dòng thứ nhất chứa ba số nguyên dương **n, k, p** lần lượt là số thí sinh trong cuộc thi, số lượng điểm số cần tính và vị trí bắt đầu tính điểm trên bảng điện tử.
- Dòng thứ hai ghi **n** số nguyên dương $a_1, a_2, ..., a_n$ ($1 \le a_i \le 10^9$).

Output: In ra một số nguyên duy nhất là kết quả của bài toán lấy dư cho $10^9 + 7$.

Subtask

- Có **40%** số test với $n \le 10^5$; $p = 1$; $k \le n$.
- Có **30%** số test với $n \le 10^5$; $p, k \le 10^6$.
- Có **30%** số test với $n \le 10^6$; $p, k \le 10^{18}$.

Example

Input	Output	Giải thích
6 7 3 4 3 6 2 9 5	35	**7** số nguyên liên tiếp xuất hiện trên màn hình bắt đầu từ số xuất hiện thứ **3** là **6 2 9 5 4 3 6**. Kết quả: **(6 + 2 + 9 + 5 + 4 + 3 + 6) % (10^9 + 7) = 35.**

Bài 13. CHIENBINH Chiến Binh

Trong một vương quốc xa xưa, một vị tướng huyền thoại đang tập hợp một đội quân bất bại để chuẩn bị cho một cuộc chiến vĩ đại. Đội quân này có một cơ chế huấn luyện đặc biệt theo quy luật sau:

- Ngày đầu tiên (ngày thứ **0**): Đội quân có **n** chiến binh ở cấp độ **1**.
- Mỗi ngày tiếp theo:
  - Mỗi chiến binh cấp **i** sẽ huấn luyện và chiêu mộ thêm **i** tân binh (tất cả đều có cấp **1**). Những tân binh này sẽ bắt đầu huấn luyện và chiêu mộ binh lính từ ngày sau.
  - Đồng thời, chiến binh cấp **i** sẽ trở nên mạnh hơn và thăng lên cấp **i + 1**.

Yêu cầu: Hãy xác định sau **k** ngày, tổng số chiến binh trong quân đội là bao nhiêu. Kết quả in ra là số nguyên duy nhất, lấy **modulo 10^9+7**.

Input: gồm hai số nguyên **n, k** ($1 \le n \le 10^5$, $1 \le k \le 10^5$).

Output: Một số nguyên duy nhất là tổng số chiến binh sau **k** ngày, lấy **modulo 10^9+7**.

Giới hạn:

- Có 40% số test với $n \le 10^2$; $k \le 10^3$
- Có 60% số test với $n \le 10^3$; $k \le 10^5$

Ví dụ:

Input	Output	Giải thích
5 4	170	Với **5** chiến binh ban đầu, sau **4** ngày tổng số chiến binh có trong quân đội là **170**.

BÀI GIẢI ĐỀ SỐ 1
(Học sinh giỏi 9 – Đà Nẵng – Năm học: 2024 – 2025)

1. Bài 1. KITU Kí tự

Cho một xâu **s** chỉ chứa các ký tự in hoa trong bảng chữ cái tiếng Anh **A...Z.**

Yêu cầu: Hãy kiểm tra xâu **s** còn thiếu những ký tự nào trong bảng chữ cái tiếng Anh **A...Z.**

Input: Một dòng duy nhất chứa xâu **s** có độ dài không quá 10^5 ký tự.

Output: In ra các ký tự chưa xuất hiện trong xâu **s**, các ký tự được viết liền tục theo thứ tự tăng dần.

Ví dụ:

Input	Output	Giải thích
EFGHJABCD	IKLMNOPQRSTUVWXYZ	

1.1. Tóm tắt đề bài:

- Đề bài yêu cầu tìm các chữ cái in hoa không có trong xâu **s** đầu vào. Sau đó ghi kết quả tìm được ra file **KITU.OUT**.

1.2. Ý tưởng giải:

- Khởi tạo một xâu bảng chữ cái **(bangchucai)** chỉ chứa 26 chữ cái in hoa tiếng Anh theo thứ tự từ A đến Z.
- Sử dụng vòng lặp **for** để duyệt qua từng chữ cái **ch** có trong xâu **bangchucai**. Nếu chữ cái **ch** không có trong xâu **s** thì thêm chữ cái **ch** này vào trong xâu kết quả (**ketqua**).
- Ghi kết quả **(ketqua)** tìm được vào file **KITU.OUT**.
- Độ phức tạp **O(n)** của cách giải trên là: O(n) = 26 x 10^5 ≈ **3 x 10^6**

1.3. Mô tả thuật toán:

- **Bước 1. Đọc dữ liệu đầu vào**
  + Mở file **KITU.INP** để đọc xâu ký tự **s** đầu vào.
  + Đóng file đầu vào **KITU.INP**
- **Bước 2. Khởi tạo bảng chữ cái**
  + Khởi tạo xâu **bangchucai** chỉ chứa 26 chữ cái in hoa tiếng Anh (A đến Z).
- **Bước 3.** Khởi tạo xâu **ketqua** = “ ” để ghi lại kết quả tìm được.
- **Bước 4. Tìm các chữ cái không xuất hiện**
  + Sử dụng vòng lặp **for** để duyệt qua từng chữ cái **ch** trong xâu **bangchucai**.
  + Nếu chữ cái **ch** đó không xuất hiện trong xâu đầu vào **(s)** thì thêm chữ cái này vào xâu **ketqua**.
- **Bước 5. Ghi kết quả ra file KITU.OUT**
  + Mở file **KITU.OUT** để ghi kết quả.
  + Ghi xâu **ketqua** vào file.
  + Đóng file đầu ra **KITU.OUT**

1.4. Code tham khảo (Python và C++):

1.4.1. Code Python:

```python
# Mo file KITU.INP de doc du lieu
fi = open('KITU.INP', 'r')
# Nhap xau s
s = fi.read()
# Dong file KITU.INP lai
fi.close()
# Khoi tao xau bangchucai gom 26 chu cai in hoa
bangchucai = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
# Khoi tao xau ketqua la xau rong de luu ket qua tim duoc
ketqua = ''
# Duyet qua tung ky tu ch trong xau bangchucai
for ch in bangchucai:
    # Neu ky tu ch khong co trong xau s thi them ch vao xau ketqua
    if ch not in s:
        ketqua = ketqua + ch
# Mo file KITU.OUT de ghi ket qua
fo = open('KITU.OUT', 'w')
# Ghi xau ketqua vao file KITU.OUT
fo.write(ketqua)
# Dong file KITU.OUT lai
fo.close()
```

1.4.2. Code C++:

```cpp
#include <iostream>
#include <bits/stdc++.h>
#include <string> // Khai bao thu vien nay de dung duoc string
#include <fstream> // Khai bao thu vien nay de dung duoc ifstream, ofstream
using namespace std;
int main()
{
    // Mo file KITU.INP de doc du lieu
    ifstream fi("KITU.INP");
    // Nhap xau s
    string s;
    fi >> s;
    // Dong file KITU.INP lai
    fi.close();
    // Khoi tao xau bangchucai gom 26 chu cai in hoa
    string bangchucai = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    // Khoi tao xau ketqua la xau rong de luu ket qua tim duoc
    string ketqua = "";
    // Duyet qua tung ky tu ch trong xau bangchucai
    for (char ch : bangchucai)
```

```
    {
        // Neu ky tu ch khong co trong xau s thi them ch vao xau ketqua
        if (s.find(ch) >= s.length())
        {
            ketqua = ketqua + ch;
        }
    }
    // Mo file KITU.OUT de ghi ket qua
    ofstream fo("KITU.OUT");
    // Ghi xau ketqua vao file KITU.OUT
    fo << ketqua;
    // Dong file KITU.OUT lai
    fo.close();
    return 0;
}
```

2. Bài 2. SOTC Số tròn chục

Số tròn chục là số có chữ số hàng đơn vị là chữ số **0**.

Cho hai số tự nhiên ***L*** và ***R***. Hãy đếm xem có bao nhiêu số tròn chục lớn hơn **L** và nhỏ hơn **R**.

Dữ liệu: Nhập vào số tự nhiên ***L, R*** **(*L* < *R* ≤ 10^{12})**. Mỗi số trên một dòng.

Kết quả: Ghi ra kết quả của bài toán.

Ví dụ:

Input	Output	Giải thích
5 31	3	Có 3 số tròn chục lớn hơn 5 và nhỏ hơn 31 là: 10, 20, 30

Chấm điểm:

- Nếu chương trình chạy đúng những trường hợp $R \le 10^6$, thí sinh sẽ được 80 điểm;
- Nếu chương trình chạy đúng những trường hợp $R \le 10^{12}$, thí sinh sẽ được 100 điểm.

2.1. Tóm tắt đề bài:

- Đề bài yêu cầu đếm số lượng các số chia hết cho 10 nằm trong khoảng giữa hai số nguyên **L** và **R (L < R, không kể L và R)**, rồi ghi kết quả ra file **SOTC.OUT**.

2.2. Ý tưởng giải:

- Tìm **số nhỏ nhất** chia hết cho 10 nhưng **lớn hơn L**. **(dau)**
- Tìm **số lớn nhất** chia hết cho 10 nhưng **nhỏ hơn R**. **(cuoi)**
- Nếu **dau > cuoi** thì không có số nào chia hết cho 10 trong khoảng **(L, R)**. Ngược lại, sử dụng công thức **(cuoi - dau) // 10 + 1** để tính số lượng các số chia hết cho 10 nằm trong khoảng **(L, R)**.
- Ghi kết quả **(ketqua)** tìm được vào file **SOTC.OUT**.

2.3. Mô tả thuật toán:

- **Bước 1. Đọc dữ liệu:**

  + Mở file **SOTC.INP**, đọc hai số nguyên **L** và **R**.

+ Đóng file đầu vào **SOTC.INP**

- Bước 2. Tìm số nhỏ nhất chia hết cho 10 nhưng lớn hơn L (dau):

+ Sử dụng công thức **dau = (L // 10) * 10**, nếu **dau ≤ L** thì **cộng** thêm 10 **(dau = dau + 10)**.

- Bước 3. Tìm số lớn nhất chia hết cho 10 nhưng nhỏ hơn R (cuoi):

+ Sử dụng công thức **cuoi = (R // 10) * 10**, nếu **cuoi ≥ R** thì **trừ** đi 10 **(cuoi = cuoi – 10)**.

- Bước 4. Kiểm tra điều kiện và tính số lượng số chia hết cho 10:

+ Nếu **dau > cuoi** thì **ketqua = 0**. Ngược lại, **ketqua = (cuoi – dau) // 10 + 1**.

- Bước 5. Ghi kết quả ra file:

+ Mở file **SOTC.OUT** và ghi kết quả vừa tính được.

+ Đóng file đầu ra **SOTC.OUT**

2.4. Code tham khảo (Python và C++):

2.4.1. Code Python:

```
# Mo file SOTC.INP de doc du lieu
fi = open('SOTC.INP', 'r')
# Nhap vao L va R
L = fi.readline()
L = int(L)
R = fi.read()
R = int(R)
# Dong file SOTC.INP
fi.close()
# Tim so nho nhat chia het cho 10 nhung lon hon L
dau = (L // 10) * 10
if dau <= L: dau = dau + 10
# Tim so lon nhat chia het cho 10 nhung nho hon R
cuoi = (R // 10) * 10
if cuoi >= R: cuoi = cuoi - 10
# Kiem tra dieu kien va tinh so luong so chia het cho 10
if dau > cuoi: ketqua = 0
else: ketqua = (cuoi - dau) // 10 + 1
# Mo file SOTC.OUT de ghi ket qua
fo = open('SOTC.OUT', 'w')
# Ghi ket qua vao file SOTC.OUT
fo.write(f'{ketqua}')
# Dong file SOTC.OUT
fo.close()
```

2.4.2. Code C++:

```
#include <iostream>
#include <bits/stdc++.h>
#include <fstream> // Khai bao thu vien nay de dung duoc ifstream, ofstream
```

```
using namespace std;
int main()
{
    // Mo file SOTC.INP de doc du lieu
    ifstream fi("SOTC.INP");
    // Doc vao 2 so L, R
    long long L, R;
    fi >> L;
    fi >> R;
    // Dong file SOTC.INP lai
    fi.close();
    // Tim so nho nhat chia het cho 10 nhung lon hon L
    long long dau = L / 10 * 10;
    if (dau <= L) dau = dau + 10;
    // Tim so lon nhat chia het cho 10 nhung nho hon R
    long long cuoi = R / 10 * 10;
    if (cuoi >= R) cuoi = cuoi - 10;
    // Kiem tra dieu kien va tinh so luong so chia het cho 10
    long long ketqua;
    if (dau > cuoi) ketqua = 0;
    else ketqua = (cuoi - dau) / 10 + 1;
    // Mo file SOTC.OUT de ghi ket qua
    ofstream fo("SOTC.OUT");
    // Ghi ket qua vao file SOTC.OUT
    fo << ketqua;
    // Dong file SOTC.OUT lai
    fo.close();
    return 0;
}
```

3. Bài 3. TONGLT Tổng liên tiếp

Trong cuộc thi "Học sinh tài năng" được tổ chức tại một trường học, ban tổ chức chuẩn bị một bảng điểm điện tử để hiển thị điểm số của từng thí sinh. Điểm số của **n** thí sinh được hiển thị theo thứ tự từ thí sinh **1** đến thí sinh **n**, sau đó lặp lại vô hạn lần. Cụ thể, sau khi hiển thị điểm của thí sinh **n**, bảng điểm sẽ quay lại hiển thị điểm của thí sinh **1** rồi thí sinh **2**, và cứ thế không ngừng.

Yêu cầu: Hãy giúp ban tổ chức tính tổng **k** điểm số liên tiếp xuất hiện trên bảng điểm, bắt đầu từ vị trí **p**.

Input

- Dòng thứ nhất chứa ba số nguyên dương **n, k, p** lần lượt là số thí sinh trong cuộc thi, số lượng điểm số cần tính và vị trí bắt đầu tính điểm trên bảng điện tử.
- Dòng thứ hai ghi **n** số nguyên dương $a_1, a_2, ..., a_n$ ($1 \le a_i \le 10^9$).

Output: In ra một số nguyên duy nhất là kết quả của bài toán lấy dư cho $10^9 + 7$.

Subtask

- Có **40%** số test với $n \le 10^3$; $p = 1$; $k \le n$.
- Có **30%** số test với $n \le 10^3$; $p, k \le 10^6$.
- Có **30%** số test với $n \le 10^6$; $p, k \le 10^{18}$.

Example

Input	Output	Giải thích
6 7 3 4 3 6 2 9 5	35	**7** số nguyên liên tiếp xuất hiện trên màn hình bắt đầu từ số xuất hiện thứ **3** là **6 2 9 5 4 3 6**. Kết quả: **(6 + 2 + 9 + 5 + 4 + 3 + 6) % (10^9 + 7) = 35.**

3.1. Tóm tắt đề bài:

- Đề bài yêu cầu:

+ Tính tổng của **k** phần tử liên tiếp trong mảng, bắt đầu từ vị trí **p**. Nếu số lượng phần tử cần lấy **k** lớn hơn độ dài mảng **n**, ta sẽ phải tính tổng của **n** phần tử trong mảng nhiều lần. Mỗi lần tính tổng của **n** phần tử trong mảng được gọi là đã tính được tổng của một vòng đầy đủ.

+ Sau khi đã tính xong tổng của các vòng đầy đủ. Nếu số lượng phần tử **k** còn lại không đủ để tạo thành vòng đầy đủ, thì sử dụng vòng lặp **for** để tính tổng của các phần tử **k** còn lại. Vòng lặp **for** này sẽ duyệt từ vị trí **p – 1 đến p – 1 + số lượng phần tử còn lại** (vị trí duyệt **for** sẽ được quay trở về vị trí đầu tiên của mảng nếu vị trí của phần tử đang duyệt vượt quá **n**).

+ Cuối cùng, cộng tổng của các vòng lặp đầy đủ với tổng của các phần tử không thể tạo thành vòng đầy đủ rồi chia dư cho 10^9 + 7 để kết quả không quá lớn.

3.2. Ý tưởng giải:

+ Tính số lần lặp đầy đủ của mảng **a** bằng cách chia **k** cho **n**: **solanlapdaydu = k // n**.

+ Tính số phần tử dư không thể tạo thành 1 lần lặp đầy đủ: **dem = k % n**

+ Tính tổng của một lần lặp đầy đủ - Tính tổng tất cả phần tử trong mảng **a**: **tong1 = sum(a)**.

+ Tính tổng giá trị của những phần tử đã tạo thành các lần lặp đầy đủ: **tong1 = tong1 * solanlapdaydu**.

+ Tính tổng giá trị của các phần tử dư không thể tạo thành một lần lặp đầy đủ:

Sử dụng vòng lặp **for** để duyệt từ **p – 1 đến (p – 1) + dem** để cộng dồn các phần tử của mảng **a** ở trong khoảng này, quay về vị trí đầu tiên của mảng **a** nếu vị trí đang tính vượt quá **n**. Rồi gán kết quả tính được vào trong biến **tong2** (Bởi vì mảng trong **Python** và **C++** được bắt đầu từ phần tử **0** nên phải lùi vị trí **p** lại 1 đơn vị so với vị trí **p** của đề bài cho)

+ Kết quả cần tìm của đề bài chính là: **ketqua = tong1 + tong2**

3.3. Mô tả thuật toán:

- Bước 1. Đọc dữ liệu từ file:

+ Mở file **TONGLT.INP** để đọc.

+ Đọc ba số nguyên **n, k, p**.

+ Đọc mảng **a** gồm **n** phần tử.

- Bước 2. Tính số lần lặp đầy đủ:

Số lần lặp đầy đủ: **solanlapdaydu = k // n**.

- Bước 3. Tính số phần tử dư không thể tạo thành 1 lần lặp đầy đủ:

Số phần tử dư: **dem = k % n**.

- Bước 4. Tính tổng của 1 lần lặp đầy đủ:

tong1 = sum(a).

- Bước 5. Tính tổng của những phần tử đã tạo thành các lần lặp đầy đủ:

tong1 = tong1 * solanlapdaydu.

- Bước 6. Tính tổng của các phần dư không thể tạo thành một lần lặp đầy đủ:

+ Khởi tạo **tong2 = 0**.

+ Duyệt từ vị trí **(p-1)** và lấy tiếp **dem** phần tử, quay về vị trí đầu tiên của mảng a nếu vị trí đang tính vượt quá **n**:

```
for i in range(0, dem):
    tong2 = (tong2 % hangso + a[(p - 1 + i) % n])
```

- Bước 7. Tính kết quả cuối cùng và ghi vào file TONGLT.OUT

+ **ketqua = (tong1 % hangso + tong2 % hangso) % hangso**.

+ Mở file **TONGLT.OUT** để ghi kết quả.

+ Ghi biến **ketqua** vào **file**.

3.4. Code tham khảo (Python và C++):

3.4.1. Code Python:

```
# Mo file TONGLT.INP de doc du lieu
fi = open('TONGLT.INP', 'r')
# Nhap cac so nguyên n, k, p
n, k, p = fi.readline().split()
```

```
n, k, p = int(n), int(k), int(p)
# Nhap mang a co n phan tu
a = fi.read().split()
a = list(map(int, a))
# Dong file TONGLT.INP sau khi doc du lieu
fi.close()
# Khoi tao hang so de chia du theo yeu cau cua de bai
hangso = 10**9 + 7
# Tinh so lan lap day du trong mang a voi k phan tu
solanlapdaydu = k // n
# Tinh so luong phan tu du khong the tao thanh 1 lan lap day du
dem = k % n
# Tinh tong cua 1 lan lap day du
tong1 = sum(a) % hangso
# Tinh tong gia tri cua nhung phan tu da tao thanh cac lan lap day du
tong1 = tong1 * (solanlapdaydu % hangso)
# Tinh tong gia tri cua cac phan tu khong the tao thanh 1 lan lap day du
tong2 = 0
# Duyet tu vi tri (p-1) va lay tiep dem phan tu, quay ve vi tri dau tien cua mang a neu
vi tri dang tinh vuot qua n
for i in range(0, dem):
    tong2 = (tong2 % hangso + a[(p - 1 + i) % n])
# Tinh ket qua theo de bai yeu cau
ketqua = tong1 % hangso + tong2 % hangso
ketqua = ketqua % hangso
# Mo file TONGLT.OUT de ghi ket qua
fo = open('TONGLT.OUT', 'w')
# Ghi ket qua vao file TONGLT.OUT
fo.write(f'{ketqua}')
# Dong file TONGLT.OUT sau khi ghi ket qua
fo.close()
```

3.4.2. Code C++:

```
#include <iostream>
#include <bits/stdc++.h>
#include <fstream> // Khai bao thu vien nay de dung duoc ifstream, ofstream
#include <vector> // Khai bao thu vien nay de dung duoc vector
using namespace std;
int main()
{
    // Mo file TONGLT.INP de doc du lieu
    ifstream fi("TONGLT.INP");
    // Nhap cac so nguyên n, k, p
    int n;
    long long k, p;
    fi >> n >> k >> p;
```

```
    // Nhap mang a co n phan tu
    vector <int> a(n);
    for (int i = 0; i < n; i++)
    {
        fi >> a[i];
    }
    // Dong file TONGLT.INP sau khi doc du lieu xong
    fi.close();
    // Khoi tao hang so de chia du theo yeu cau cua de bai
    const long long hangso = 1000000007;
    // Tinh so lan lap day du trong mang a voi k phan tu
    long long solanlapdaydu = k / n;
    // Tinh so luong phan tu du khong the tao thanh 1 lan lap day du
    int dem = k % n;
    // Tinh tong cua 1 lan lap day du
    long long tong1 = 0;
    for (int i = 0; i < n; i++)
    {
        tong1 = (tong1 + a[i]) % hangso;
    }
    // Tinh tong gia tri cua nhung phan tu da tao thanh cac lan lap day du
    tong1 = tong1 * (solanlapdaydu % hangso);
    // Tinh tong gia tri cua cac phan tu khong the tao thanh 1 lan lap day du
    long long tong2 = 0;
    // Duyet tu vi tri (p-1) va lay tiep dem phan tu, quay ve vi tri dau tien cua mang
a neu vi tri dang tinh vuot qua n
    for (int i = 0; i < dem; i++)
    {
        tong2 = (tong2 % hangso + a[(p - 1 + i) % n]);
    }
    // Tinh ket qua theo de bai yeu cau
    long long ketqua = tong1 % hangso + tong2 % hangso;
    ketqua = ketqua % hangso;
    // Mo file TONGLT.OUT de ghi ket qua
    ofstream fo("TONGLT.OUT");
    // Ghi ket qua vao file TONGLT.OUT
    fo << ketqua;
    // Dong file TONGLT.OUT sau khi ghi ket qua xong
    fo.close();
    return 0;
}
```

4. Bài 4. CHIENBINH Chiến binh

Trong một vương quốc xa xưa, một vị tướng huyền thoại đang tập hợp một đội quân bất bại để chuẩn bị cho một cuộc chiến vĩ đại. Đội quân này có một cơ chế huấn luyện đặc biệt theo quy luật sau:

- Ngày đầu tiên (ngày thứ **0**): Đội quân có **n** chiến binh ở cấp độ **1**.
- Mỗi ngày tiếp theo:
  - Mỗi chiến binh cấp **i** sẽ huấn luyện và chiêu mộ thêm **i** tân binh (tất cả đều có cấp **1**). Những tân binh này sẽ bắt đầu huấn luyện và chiêu mộ binh lính từ ngày sau.
  - Đồng thời, chiến binh cấp **i** sẽ trở nên mạnh hơn và thăng lên cấp **i + 1**.

Yêu cầu: Hãy xác định sau **k** ngày, tổng số chiến binh trong quân đội là bao nhiêu. Kết quả in ra là số nguyên duy nhất, lấy **modulo** 10^9+7.

Input: gồm hai số nguyên **n, k** ($1 \le n \le 10^3$, $1 \le k \le 10^5$).

Output: Một số nguyên duy nhất là tổng số chiến binh sau **k** ngày, lấy **modulo** 10^9+7.

Giới hạn:

- Có 40% số test với $n \le 10^2$; $k \le 10^3$
- Có 60% số test với $n \le 10^3$; $k \le 10^5$

Ví dụ:

Input	Output	Giải thích
5 4	170	Với **5** chiến binh ban đầu, sau **4** ngày tổng số chiến binh có trong quân đội là **170**.

4.1. Tóm tắt đề bài:

- Ban đầu: Có **n** chiến binh, tất cả **n** chiến binh này đều là chiến binh **cấp 1 (i=1)**.

- Sau mỗi **k** ngày: Mỗi chiến binh **cấp i** của ngày hôm trước, sẽ sinh ra thêm được **i** chiến binh **cấp 1**. Sau khi đã sinh ra xong các chiến binh cấp 1, mỗi chiến binh **cấp i** của ngày hôm trước sẽ **tăng thêm 1 cấp** và trở thành cấp **i + 1**.

- Hỏi sau **k** ngày, **tổng số** chiến binh có được sẽ là bao nhiêu?

4.2. Ý tưởng giải:

- Để giải được bài này ta cần phải hiểu được ý nghĩa của bảng sau:

Trong bảng này:

k: Thứ tự ngày

x1 (Cấp 1), **x2** (Cấp 2), **x3** (Cấp 3), …: Số lượng chiến binh từng cấp có được trong ngày

n: Tổng số chiến binh có được trong ngày **(n = x1 + x2 + x3 + …)**

Ngày (k)	**Số lượng chiến binh**						**Tổng số chiến binh (n)**
	x1	**x2**	**x3**	**x4**	**x5**	**x6**	
0	5						**5**
1	5	5					**10**
2	15	5	5				**25**
3	40	15	5	5			**65**
4	105	40	15	5	5		**170**
5	275	105	40	15	5	5	**445**
…	…	…	…	…	…	…	**…**

- Giải thích bảng trên:

+ **Ngày 0:** Có **5 (x1)** => Như vậy, sau Ngày 0 ta có được: **5 (x1)** => **n = 5**

+ **Ngày 1:** **5 (x1)** của Ngày 0 tạo ra: **5 (x1)** trong Ngày 1 và trở thành **5 (x2)** trong Ngày 1 => Như vậy, sau Ngày 1 ta có được: **5 (x1), 5 (x2)** => **n = 5 + 5 = 10**

+ **Ngày 2:**

▪ **5 (x1)** của Ngày 1 tạo ra: **5 (x1)** trong Ngày 2 và trở thành **5 (x2)** trong ngày 2

▪ **5 (x2)** của Ngày 1 tạo ra: **10 (x1)** trong Ngày 2 và trở thành **5 (x3)** trong ngày 2

=> Như vậy, sau Ngày 2 ta có được: **15 (x1), 5 (x2), 5 (x3)**

=> **n = 15 + 5 + 5 = 25**

+ **Ngày 3:**

▪ **15 (x1)** của Ngày 2 tạo ra: **15 (x1)** trong Ngày 3 và trở thành **15 (x2)** trong ngày 3

▪ **5 (x2)** của Ngày 2 tạo ra: **10 (x1)** trong Ngày 3 và trở thành **5 (x3)** trong ngày 3

▪ **5 (x3)** của Ngày 2 tạo ra: **15 (x1)** trong Ngày 3 và trở thành **5 (x4)** trong ngày 3

=> Như vậy, sau Ngày 3 ta có được: **40 (x1), 15 (x2), 5 (x3), 5 (x4)**

=> **n = 40 + 15 + 5 + 5 = 65**

+ **Ngày 4:**

▪ **40 (x1)** của Ngày 3 tạo ra: **40 (x1)** trong Ngày 4 và trở thành **40 (x2)** trong ngày 4

▪ **15 (x2)** của Ngày 3 tạo ra: **30 (x1)** trong Ngày 4 và trở thành **15 (x3)** trong ngày 4

▪ **5 (x3)** của Ngày 3 tạo ra: **15 (x1)** trong Ngày 4 và trở thành **5 (x4)** trong ngày 4

▪ **5 (x4)** của Ngày 3 tạo ra: **20 (x1)** trong Ngày 4 và trở thành **5 (x5)** trong ngày 4

=> Như vậy, sau Ngày 4 ta có được:

105 (x1), 40 (x2), 15 (x3), 5 (x4), 5 (x5)

=> n = 105 + 40 + 15 + 5 + 5 = 170

+ **Ngày 5:**

▪ **105 (x1)** của Ngày 4 tạo ra: **105 (x1)** trong Ngày 5 và trở thành **105 (x2)** trong ngày 5

▪ **40 (x2)** của Ngày 4 tạo ra: **80 (x1)** trong Ngày 5 và trở thành **40 (x3)** trong ngày 5

▪ **15 (x3)** của Ngày 4 tạo ra: **45 (x1)** trong Ngày 5 và trở thành **15 (x4)** trong ngày 5

▪ **5 (x4)** của Ngày 4 tạo ra: **20 (x1)** trong Ngày 5 và trở thành **5 (x5)** trong ngày 5

▪ **5 (x5)** của Ngày 4 tạo ra: **25 (x1)** trong Ngày 5 và trở thành **5 (x6)** trong ngày 5

=> Như vậy, sau Ngày 5 ta có được:

275 (x1), 105 (x2), 40 (x3), 15 (x4), 5 (x5), 5 (x6)

=> n = 275 + 105 + 40 + 15 + 5 + 5 = 445

+ Khi nhìn vào bảng ta thấy, **cột n** gồm có các số: **5; 10; 25; 65; 170; 445; …**

+ Ta gọi **dãy các số n** ở trên là **dãy a**, các số trong **dãy a** có quy luật như sau:

a[0] = 5

a[1] = 10

a[2] = 25 => a[2] = 3 * a[1] – a[0]

a[3] = 65 => a[3] = 3 * a[2] – a[1]

…

a[k] = 3 * a[k – 1] – a[k – 2]

+ Từ quy luật trên ta thấy, từ ngày thứ **k = 2 trở đi** thì:

a[k] = 3 * a[k – 1] – a[k – 2] (Đây chính là công thức truy hồi để tính giá trị của các phần tử còn lại trong mảng a)

a[k] chính là **kết quả cần tìm** theo yêu cầu của đề bài

4.3. Mô tả thuật toán:

- **Bước 1. Đọc dữ liệu từ file CHIENBINH.INP:**

+ Mở file **CHIENBINH.INP** để đọc.

+ Đọc 2 số nguyên **n, k**.

- **Bước 2.** Khởi tạo **mảng a**:

+ a = []

+ a[0] = a.append(n)

+ a[1] = a.append(n + n)

- **Bước 3.** Gán giá trị **k** ban đầu vào biến **n** để dùng cho vòng lặp **for**, vì không cần dùng đến **n** nữa.

+ n = k

- **Bước 4.** Tính các phần tử **k** còn lại của **mảng a** từ **a[2]** đến **a[n]** theo công thức truy hồi: **a[k] = 3 * a[k – 1] – a[k – 2].** Với **k** chạy từ **2** đến **n (n** chính là **k** ban đầu**)**

- **Bước 5.** Ghi kết quả **a[k]** ra file **CHIENBINH.OUT**.

4.4. Code tham khảo (Python và C++):

4.4.1. Code Python:

```
# Mo file de doc du lieu
fi = open('CHIENBINH.INP', 'r')
# Nhap n, k tu file
```

```
n = fi.readline()
n = int(n)
k = fi.read()
k = int(k)
# Dong file sau khi doc du lieu
fi.close()
# Khoi tao hang so de chia du theo yeu cau cua de bai
hangso = 1000000007;
# Khoi tao mang a de luu cac gia tri a[k]
a = []
# Gan a[0], a[1] ban dau vao mang a
# a[0] = n
a.append(n)
# a[1] = n + n
a.append(n + n)
# Gan gia tri k vao bien n de su dung trong vong lap for, vi khong can dung den n nua
n = k
# Tinh cac a[k] (Voi k chay tu 2 den n, n chinh la k ban dau)
for k in range(2, n + 1):
    # Cong thuc truy hoi de tinh phan tu thu k: a[k] = 3 * a[k - 1] - a[k - 2]
    ak = (3 * a[k - 1] - a[k - 2]) % hangso
    # Them gia tri a[k] vao mang a
    a.append(ak)
# Mo file de ghi ket qua
fo = open('CHIENBINH.OUT', 'w')
# Ghi ket qua ra file
fo.write(f'{a[k]}')
# Dong file sau khi ghi ket qua
fo.close()
```

4.4.2. Code C++:

```
#include <iostream>
#include <bits/stdc++.h>
#include <fstream> // Khai bao thu vien nay de dung duoc ifstream, ofstream
#include <vector> // Khai bao thu vien nay de dung duoc vector
using namespace std;
int main()
{
    // Mo file de doc du lieu
    ifstream fi("CHIENBINH.INP");
    // Nhap n, k tu file
    int n, k;
    fi >> n >> k;
    // Dong file sau khi doc du lieu
    fi.close();
    // Khoi tao hang so de chia du theo yeu cau cua de bai
```

```
    long long hangso = 1000000007;
    // Khoi tao mang a de luu cac gia tri a[k]
    vector <long long> a;
    // Gan a[0], a[1] ban dau vao mang a
    // a[0] = n
    a.push_back(n % hangso);
    // a[1] = n + n
    a.push_back((n + n) % hangso);
    // Gan gia tri k ban dau vao bien n de su dung trong vong lap for, vi khong can dung
den n nua
    n = k;
    // Tinh cac a[k] (Voi k chay tu 2 den n, n chinh la k ban dau)
    for (int k = 2; k < n + 1; k++)
    {
        // Cong thuc truy hoi de tinh phan tu thu k: a[k] = 3 * a[k - 1] - a[k - 2] co
+ hangso de tranh truong hop am
        long long ak = (3LL* a[k - 1] % hangso - a[k - 2] + hangso)
% hangso;
        // Them gia tri a[k] vao mang a
        a.push_back(ak);
    }
    // Mo file de ghi ket qua
    ofstream fo("CHIENBINH.OUT");
    // Ghi ket qua vao file
    fo << a[k] << endl;
    // Dong file sau khi ghi ket qua
    fo.close();
    return 0;
}
```

5. Các bộ Test:

Mỗi câu của đề thi đều có đủ 10 bộ Test, với các giá trị đầu vào tương ứng với tỉ lệ (%) của các ràng buộc (Substack) mà đề bài yêu cầu. Những bộ Test này đã được gửi ở tệp tin đính kèm cùng với tài liệu này.

6. Kết quả chấm thử bằng phần mềm Themis 1.9, giới hạn thời gian chạy của mỗi bộ Test là 01 giây:

Tên bài	Điểm tối đa	Điểm chấm bằng Themis 1.9	
		Python	C++
Bài 1. Kí tự (KITU.*)	100 điểm	100 điểm (100%)	100 điểm (100%)
Bài 2. Số tròn chục (SOTC.*)	100 điểm	100 điểm (100%)	100 điểm (100%)
Bài 3. Tổng liên tiếp (TONGLT.*)	100 điểm	100 điểm (100%)	100 điểm (100%)
Bài 4. Chiến binh (CHIENBINH.*)	100 điểm	100 điểm (100%)	100 điểm (100%)
Tổng số điểm đạt được:		**400 điểm (100%)**	**400 điểm (100%)**

ĐỀ SỐ 2
(Học sinh giỏi 9 – Đà Nẵng – Năm học: 2023 – 2024)

SỞ GIÁO DỤC VÀ ĐÀO TẠO
THÀNH PHỐ ĐÀ NẴNG

KÌ THI CHỌN HỌC ...
NĂM HỌC 2023 - 2024
MÔN: TIN HỌC

ĐỀ CHÍNH THỨC

Thời gian làm bài: 150 phút *(không kể thời gian giao đề)*
(Đề thi có 02 trang)

TỔNG QUAN

Câu	Tên bài	File chương trình	Dữ liệu vào	Dữ liệu ra	Điểm
1	TÍNH TỔNG	TONG.*	Từ bàn phím	In ra màn hình	3.0
2	OẢN TÙ XÌ	GAME.*	GAME.INP	GAME.OUT	3.0
3	CHÚM ĐÊN	CHUMDEN.*	CHUMDEN.INP	CHUMDEN.OUT	2.0
4	TẶNG QUÀ	TANGQUA.*	TANGQUA.INP	TANGQUA.OUT	2.0

Chú ý: Dấu * có thể là pas, c, cpp hoặc py tùy theo ngôn ngữ và môi trường lập trình.

Câu 1. TÍNH TỔNG

Cho một dãy số nguyên dương có N phần tử và một chỉ số K. Hãy tính tổng K phần tử lớn nhất trong dãy số nguyên dương đã cho

Dữ liệu vào: Đọc từ file văn bản TONG.INP
- Dòng đầu tiên chứa hai số nguyên dương N và K.
- Dòng thứ 2 chứa N số nguyên dương lần lượt là giá trị các phần tử trong dãy số.

Dữ liệu ra: Ghi vào file văn bản TONG.OUT số nguyên theo yêu cầu đề bài.

Ví dụ

TONG.INP	TONG.OUT
10 3 1 2 3 4 5 6 7 8 9 10	27

Ràng buộc:
- Có 40% test tương ứng với $K = 2, N \le 10$;
- Có 30% test tương ứng với $K = 3, N \le 100$;
- Có 30% test tương ứng với $N \le 10^5$.

Câu 2. OẢN TÙ XÌ

Nhân dịp tết cổ truyền Đức và Nhi được bố mẹ cho rất nhiều kẹo, vì được nghỉ học nên Đức và Nhi bày ra một trò chơi như sau. Hai bạn chơi oẳn tù xì với nhau, ai thắng có thể lấy 1 viên kẹo, để ghi lại kết quả Đức sử dụng các kí tự để ghi chú, nếu Đức thắng sẽ dùng kí tự D nếu Nhi thắng sẽ dùng kí tự N, nếu hoà sẽ dùng kí tự H.

Yêu cầu: Hãy cho biết số lượng kẹo của Đức và Nhi là bao nhiêu sau khi kết thúc trò chơi.

Dữ liệu vào: Đọc từ file văn bản GAME.INP chuỗi kí tự dùng để ghi lại kết quả.

Dữ liệu ra: Ghi vào file văn bản GAME.OUT 2 số nguyên lần lượt là số kẹo của Đức và Nhi.

Ví dụ

GAME.INP	GAME.OUT	Giải thích
HDNDNNNDDNN	4 6	Hai bạn chơi 11 ván trong đó Đức thắng 4 ván, Nhi thắng 6 ván và 1 ván hòa

Ràng buộc:
- Có 40% Chuỗi kí tự có độ dài tối đa 200 kí tự;
- Có 60% Chuỗi kí tự có độ dài tối đa 10^3 kí tự.

Câu 3. CHÙM ĐÈN

Dọc theo khu vườn nhà Tí được trang trí bởi n chùm bóng đèn $a_1, a_2, \dots, a_n$, chùm đèn thứ i có a_i bóng. Để chuẩn bị cho buổi tiệc sinh nhật của mình, Tí quyết định chọn một dãy các chùm đèn liên tiếp trong khu vườn để bố trí khu vực chụp hình cho khách mời đến dự tiệc. Ngoài ra, để khu vực chụp hình không quá đơn điệu, Tí muốn dãy các chùm đèn được chọn có đúng k chùm đèn có số bóng là số lẻ. Vì có quá nhiều cách chọn, Tí đang rất phân vân không biết nên chọn như thế nào.

Bạn là một trong những người bạn thân của Tí, hãy giúp Tí đếm số cách chọn các chùm đèn trong khu vườn thỏa mãn yêu cầu Tí đặt ra.

Dữ liệu vào: Đọc từ tệp văn bản CHUMDEN.INP

- Dòng thứ nhất chứa hai số nguyên n và $k (1 \le k \le n \le 10^6)$.
- Dòng thứ hai chứa n số nguyên $a_1, a_2, \dots, a_n (1 \le a_i \le 10^6)$ các số cách nhau một khoảng trắng.

Dữ liệu ra: Ghi ra tệp văn bản CHUMDEN.OUT một số nguyên duy nhất là kết quả của bài toán.

Ví dụ:

CHUMDEN.INP	CHUMDEN.OUT	Giải thích
4 2 1 3 2 3	3	Có 3 cách chọn thỏa mãn ở các vị trí bắt đầu và kết thúc là (1, 2); (1, 3) và (2, 4)

Ràng buộc:

- Có 30% số test đầu với $1 \le n \le 100$;
- Có 30% số test tiếp theo với $100 < n \le 5.10^3$;
- Có 40% số test còn lại không có ràng buộc gì thêm.

Câu 4. TẶNG QUÀ

Bố Tí là một người rất giàu có, ông có rất nhiều đất đai và các món đồ quý hiếm. Đặc biệt ông có bộ sưu tập gồm n món đồ cổ được đánh số thứ tự từ 1 đến n có giá trị cao. Ông đã nhờ các chuyên gia về đồ cổ định giá cho từng món đồ cổ của mình. Sau khi định giá các chuyên gia đã đưa ra giá trị của món đồ cổ thứ i là $a_i (\forall i = 1..n)$. Tí là đứa con duy nhất nên ông đã quyết định tặng cho Tí một số món từ bộ sưu tập đồ cổ của mình để làm vốn riêng. Ông cho Tí được tự ý lựa chọn các món đồ tuy nhiên có một yêu cầu cho Tí là các món chọn sau phải có số thứ tự và giá trị cao hơn món chọn trước đó.

Yêu cầu: Hãy giúp Tí tính xem phải chọn những món đồ trong bộ sưu tập đồ cổ như thế nào để số món đồ không được chọn là ít nhất.

Dữ liệu vào: Đọc từ tệp văn bản TANGQUA.INP

- Dòng thứ nhất chứa số nguyên dương $n (n \le 10^5)$ là số lượng các món đồ cổ.
- Dòng thứ hai ghi n số nguyên $a_1, a_2, \dots, a_n (1 \le a_i \le 10^9)$ là giá trị của từng món đồ.

Dữ liệu ra: Ghi ra tệp văn bản TANGQUA.OUT một số nguyên duy nhất là số món đồ cổ Tí không chọn.

Ví dụ:

TANGQUA.INP	TANGQUA.OUT	Giải thích
5 1 3 3 2 8	2	Tí chọn được 3 món đồ trong 5 món ở các vị trí lần lượt là (1, 3, 5)

Ràng buộc:

- Có 40% số test đầu với $n \le 25$;
- Có 30% số test tiếp theo với $25 < n \le 2000$;
- Có 30% số test còn lại không có ràng buộc gì thêm.

BÀI GIẢI ĐỀ SỐ 2
(Học sinh giỏi 9 – Đà Nẵng – Năm học: 2023 – 2024)

1. Câu 1. TÍNH TỔNG

Cho một dãy số nguyên dương có N phần tử và một chỉ số K. Hãy tính tổng K phần tử lớn nhất trong dãy số nguyên dương đã cho

Dữ liệu vào: Đọc từ file văn bản TONG.INP
- Dòng đầu tiên chứa hai số nguyên dương N và K.
- Dòng thứ 2 chứa N số nguyên dương lần lượt là giá trị các phần tử trong dãy số.

Dữ liệu ra: Ghi vào file văn bản TONG.OUT số nguyên theo yêu cầu đề bài.

Ví dụ

TONG.INP	TONG.OUT
10 3 1 2 3 4 5 6 7 8 9 10	27

Ràng buộc:
- Có 40% test tương ứng với $K = 2, N \leq 10$;
- Có 30% test tương ứng với $K = 3, N \leq 100$;
- Có 30% test tương ứng với $N \leq 10^5$.

1.1. Tóm tắt đề bài:

- Đề bài cho một mảng **a** có **N** phần tử và một số **K**. Hãy tính tổng **K** phần tử lớn nhất trong mảng **a** đã cho.

1.2. Ý tưởng giải:

- Sắp xếp mảng **a** theo thứ tự giảm dần.
- Dùng vòng lặp **for** duyệt từ đầu mảng **a** đến **K** (Duyệt mảng từ **a[0]** .. **a[K]**) rồi cộng dồn các phần tử vừa duyệt ta được tổng theo yêu cầu.

1.3. Mô tả thuật toán:

- **Bước 1. Đọc dữ liệu**
  + Mở file **TONG.INP** để nhập dữ liệu đầu vào
  + Nhập **N, K**
  + Nhập mảng **a**
- **Bước 2. Tiền xử lí**
  + Sắp xếp mảng **a** theo thứ tự giảm dần
  + Khởi tạo biến **tong = 0** để ghi lại giá trị của tổng từ **0** đến **K – 1** trên mảng **a** (**K** phần tử)
- **Bước 4. Tính tổng theo yêu cầu của đề bài**
  + Sử dụng vòng lặp **for** để duyệt mảng **a** từ **0** đến **K – 1** (**K** phần tử)
  + Cộng dồn các phần tử đang duyệt trên mảng **a** lại với nhau để ra kết quả
- **Bước 5. Ghi kết quả vào file TONG.INP**

1.4. Code tham khảo (Python và C++):

1.4.1. Code Python:

```
# Mo file de doc
fi = open('TONG.INP', 'r')
# Nhap gia tri N, K
N, K = fi.readline().split()
N, K = int(N), int(K)
```

```
# Nhap mang a
a = fi.read().split()
a = list(map(int, a))
# Dong file
fi.close()
# Sap xep mang a giam dan
a.sort(reverse=True)
# Khoi tao bien tong ban dau tong = 0 de ghi lai gia tri cua tong tu 0 den K - 1 (K
phan tu) trong mang a
tong = 0
# Duyet vong lap for de cong don tong tu 0 den K - 1 (K phan tu - Vi mang la bat dau tu
phan tu a[0])
for i in range(0, K): tong = tong + a[i]
# Mo file de ghi
fo = open('TONG.OUT', 'w')
# Ghi ket qua vao file
fo.write(f'{tong}')
# Dong file
fo.close()
```

1.4.2. Code C++:

```
#include <iostream>
#include <bits/stdc++.h>
#include <fstream> // Khai bao cai nay de su dung ifstream va ofstream
#include <vector> // Khai bao cai nay de su dung vector
using namespace std;
int main()
{
    // Mo file de doc
    ifstream fi("TONG.INP");
    // Nhap gia tri N, K
    int N, K;
    fi >> N >> K;
    // Nhap mang a
    vector <int> a(N);
    for (int i = 0; i < N; i++) fi >> a[i];
    // Dong file
    fi.close();
    // Sap xep mang a giam dan
    sort(a.begin(), a.end(), greater<int>());
    // Khoi tao bien tong ban dau tong = 0 de ghi lai gia tri cua tong tu 0 den K - 1
(K phan tu) trong mang a
    long long tong = 0;
    // Duyet vong lap for de cong don tong tu 0 den K - 1 (K phan tu - Vi mang la bat
dau tu phan tu a[0])
    for (int i = 0; i < K; i++) tong = tong + a[i];
    // Mo file de ghi
    ofstream fo("TONG.OUT");
```

```
    // Ghi ket qua vao file
    fo << tong;
    // Dong file
    fo.close();
    return 0;
}
```

2. Câu 2. OẴN TÙ TÌ

Nhân dịp tết cổ truyền Đức và Nhi được bố mẹ cho rất nhiều kẹo, vì được nghỉ học nên Đức và Nhi bày ra một trò chơi như sau. Hai bạn chơi oẳn tù xì với nhau, ai thắng có thể lấy 1 viên kẹo, để ghi lại kết quả Đức sử dụng các kí tự để ghi chú, nếu Đức thắng sẽ dùng kí tự D nếu Nhi thắng sẽ dùng kí tự N, nếu hoà sẽ dùng kí tự H.

Yêu cầu: Hãy cho biết số lượng kẹo của Đức và Nhi là bao nhiêu sau khi kết thúc trò chơi.

Dữ liệu vào: Đọc từ file văn bản GAME.INP chuỗi kí tự dùng để ghi lại kết quả.
Dữ liệu ra: Ghi vào file văn bản GAME.OUT 2 số nguyên lần lượt là số kẹo của Đức và Nhi.

Ví dụ

GAME.INP	GAME.OUT	Giải thích
HDNDNNNDDNN	4 6	Hai bạn chơi 11 ván trong đó Đức thắng 4 ván, Nhi thắng 6 ván và 1 ván hòa

Ràng buộc:
- Có 40% Chuỗi kí tự có độ dài tối đa 200 kí tự;
- Có 60% Chuỗi kí tự có độ dài tối đa 10^3 kí tự.

2.1. Tóm tắt đề bài:

Cho một **mảng s** chỉ chứa các phần tử **D (hoặc d), N (hoặc n), H (hoặc h)**. Hãy đếm xem trong mảng **s** đã cho có bao nhiêu phần tử **D (hoặc d)**, có bao nhiêu phần tử **N (hoặc n)**. Với số lượng phần tử tối đa của mảng **s** là **10^3 (len(s) ≤ 10^3)**.

2.2. Ý tưởng giải:

- Khởi tạo các biến **demD = 0, demN = 0** để lưu lại số lần xuất hiện của các kí tự **D** và kí tự **N** có trong mảng **s**
- Tính số lần xuất hiện của các kí tự **D** và kí tự **N** trong mảng **s** bằng cách sử dụng vòng lặp **for** để duyệt từ đầu mảng **s** đến cuối mảng **s**:
  + Nếu kí tự đang xét **s[i] = 'D'** thì **tăng** biến **demD** lên **1** đơn vị
  + Nếu kí tự đang xét **s[i] = 'N'** thì **tăng** biến **demN** lên **1** đơn vị

2.3. Mô tả thuật toán:

- **Bước 1.** Đọc dữ liệu từ file **GAME.INP**
- **Bước 2.** Khởi tạo các biến **demD = 0, demN = 0** để lưu lại số lần xuất hiện của các kí tự **D** và kí tự **N** có trong mảng **s**
- **Bước 3.** Tính số lần xuất hiện của các kí tự **D** và kí tự **N** trong mảng **s**
  + Sử dụng vòng lặp **for** để duyệt từ đầu mảng **s** đến cuối mảng **s**:
    - Nếu kí tự đang xét **s[i] = 'D'** thì **tăng** biến **demD** lên **1** đơn vị
    - Nếu kí tự đang xét **s[i] = 'N'** thì **tăng** biến **demN** lên **1** đơn vị
- **Bước 4.** Ghi kết quả **demD, demN** vừa tính được ở trên ra file **GAME.OUT**

2.4. Code tham khảo (Python và C++):

2.4.1. Code Python:

```
# Mo file de doc
fi = open('GAME.INP', 'r')
```

```
# Khoi tao mang s de doc tung ki tu co trong file GAME.INP
s = []
# Doc toan bo noi dung file GAME.INP va dua tung ki tu si vao trong mang s
for si in fi.read():
    # Bo qua cac khoang trang, xuong dong, tab, ... cua du lieu trong file GAME.INP
    if not si.isspace():
        # Them phan tu si vao trong mang s
        s.append(si)
# Dong file
fi.close()
# Khoi tao cac bien demD, demN de luu lai so lan xuat hien cua ki tu D va ki tu N co
trong mang s
demD = 0
demN = 0
# Duyet tu dau mang s den cuoi mang s de cong don cac gia tri demD, demN
for i in range(0, len(s)):
    if s[i] == 'D' or s[i] == 'd': demD = demD + 1
    if s[i] == 'N' or s[i] == 'n': demN = demN + 1
# Mo file de ghi
fo = open('GAME.OUT', 'w')
# Ghi ket qua ra file
fo.write(f'{demD} {demN}')
# Dong file
fo.close()
```

2.4.2. Code C++:

```
#include <iostream>
#include <bits/stdc++.h>
#include <fstream> // Khai bao cai nay de su dung ifstream va ofstream
#include <vector> // Khai bao cai nay de su dung vector
using namespace std;
int main()
{
    // Mo file de doc
    ifstream fi("GAME.INP");
    // Khoi tao mang s de doc tung ki tu co trong file GAME.INP
    vector <char> s;
    // Doc toan bo noi dung file GAME.INP va dua tung ki tu si vao trong mang s
    char si;
    while (fi >> si) s.push_back(si); // Them phan tu si vao trong mang s
    // Dong file
    fi.close();
    // Khoi tao cac bien demD, demN de luu lai so lan xuat hien cua ki tu D va ki tu N
co trong mang s
    int demD = 0;
    int demN = 0;
    // Duyet tu dau mang s den cuoi mang s de cong don cac gia tri demD, demN
    for (int i = 0; i < s.size(); i++)
```

```
    {
        if (s[i] == 'D' || s[i] == 'd') demD = demD + 1;
        if (s[i] == 'N' || s[i] == 'n') demN = demN + 1;
    }
    // Mo file de ghi
    ofstream fo("GAME.OUT");
    // Ghi ket qua ra file
    fo << demD << " " << demN;
    // Dong file
    fo.close();
    return 0;
}
```

3. Câu 3. CHÙM ĐÈN

Câu 3. CHÙM ĐÈN

Dọc theo khu vườn nhà Tí được trang trí bởi n chùm bóng đèn $a_1, a_2, \dots, a_n$, chùm đèn thứ i có a_i bóng. Để chuẩn bị cho buổi tiệc sinh nhật của mình, Tí quyết định chọn một dãy các chùm đèn liên tiếp trong khu vườn để bố trí khu vực chụp hình cho khách mời đến dự tiệc. Ngoài ra, để khu vực chụp hình không quá đơn điệu, Tí muốn dãy các chùm đèn được chọn có đúng k chùm đèn có số bóng là số lẻ. Vì có quá nhiều cách chọn, Tí đang rất phân vân không biết nên chọn như thế nào.

Bạn là một trong những người bạn thân của Tí, hãy giúp Tí đếm số cách chọn các chùm đèn trong khu vườn thỏa mãn yêu cầu Tí đặt ra.

Dữ liệu vào: Đọc từ tệp văn bản CHUMDEN.INP

- Dòng thứ nhất chứa hai số nguyên n và $k (1 \le k \le n \le 10^6)$.
- Dòng thứ hai chứa n số nguyên $a_1, a_2, \dots, a_n (1 \le a_i \le 10^6)$ các số cách nhau một khoảng trắng.

Dữ liệu ra: Ghi ra tệp văn bản CHUMDEN.OUT một số nguyên duy nhất là kết quả của bài toán.

Ví dụ:

CHUMDEN.INP	CHUMDEN.OUT	Giải thích
4 2 1 3 2 3	3	Có 3 cách chọn thỏa mãn ở các vị trí bắt đầu và kết thúc là (1, 2); (1, 3) và (2, 4)

Ràng buộc:

- Có 30% số test đầu với $1 \le n \le 100$;
- Có 30% số test tiếp theo với $100 < n \le 5.10^3$;
- Có 40% số test còn lại không có ràng buộc gì thêm.

3.1. Tóm tắt đề bài:

- Cho 2 số nguyên dương **N, K** và một mảng **a** chứa **N** số nguyên dương. Hãy cho biết trong mảng **a** có nhiều nhất bao nhiêu mảng con chứa đúng **K** số lẻ.

3.2. Ý tưởng giải: (Sẽ đưa ra ví dụ cụ thể mô tả từng bước để giải)

- Dữ liệu vào:

N = 13, K = 3

a = [2, 1, 4, 3, 6, 5, 7, 8, 9, 10, 11, 12, 13]

- Xử lí:

+ Chèn thêm số **0** vào đầu mảng **a** để có thể bắt đầu mảng tại **vị trí 1** cho dễ hiểu, thay vì **vị trí 0** như mặc định.

=> a = [0, 2, 1, 4, 3, 6, 5, 7, 8, 9, 10, 11, 12, 13]

+ Tìm vị trí các số lẻ xuất hiện trong mảng **a** rồi gán vào mảng **b**:

- b = [2, 4, 6, 7, 9, 11, 13]
- Số lượng số lẻ trong mảng **a** là: **m = len(b) = 7**

▪ Có **K = 3** => Công thức tính số lượng đoạn con chứa đúng **K** phần tử lẻ trong mảng **b** là: **m – K + 1 = 7 – 3 + 1 = 5**

▪ Chèn thêm số **0** vào đầu mảng b và chèn thêm **số N + 1** vào sau mảng **b** để có thể tính được theo công thức **Left, Right** sắp trình bày ở phía dưới => b = [0, 2, 4, 6, 7, 9, 11, 13, 14] (**N = 13** theo ví dụ => **N + 1** = 13 + 1 = **14**)

+ 5 đoạn con chứa K phần tử lẻ trong mảng **b** nằm tại các vị trí cụ thể trong mảng **a** như sau:

▪ Đoạn 1: [2, 4, 6]
▪ Đoạn 2: [4, 6, 7]
▪ Đoạn 3: [6, 7, 9]
▪ Đoạn 4: [7, 9, 11]
▪ Đoạn 5: [9, 11, 13]

+ Xét lần lượt từng đoạn con, ở đây sẽ xét Đoạn 2, Đoạn 3, Đoạn 4 trước để dễ hình dung công thức tính. Còn Đoạn 1 và Đoạn 5 sẽ xét sau:

▪ Đoạn 2: [4, 6, 7]

◦ Tính Left = Phần tử đầu của Đoạn 2 – Phần tử liền kề trước nó

=> Vì phần tử đầu tiên của **Đoạn 2** nằm ở vị trí thứ **2** của mảng **b** và có giá trị bằng **4**, nên: **Left = b[2] – b[1] = 4 – 2 = 2**

◦ Tính Right = Phần tử đứng liền sau phần tử cuối cùng của Đoạn 2 – Phần tử cuối cùng của Đoạn 2

=> Vì phần tử cuối cùng của **Đoạn 2** nằm ở vị trí thứ **4** của mảng **b** và có giá trị bằng **7**, nên: **Right = b[4 + 1] – b[4] = b[5] – b[4] = 9 – 7 = 2**

◦ Tổng số cách khác nhau được chọn theo yêu cầu của đề bài là: **tongsocach = Left * Right = 2 * 2 = 4** (Theo nguyên lí nhân: Mỗi điểm bắt đầu **Left** đều kết hợp được với mọi điểm kết thúc **Right** để tạo thành một đoạn con hợp lệ).

▪ Đoạn 3: [6, 7, 9] (tương tự như cách tính của Đoạn 2)

◦ Left = b[3] – b[2] = 6 – 4 = 2
◦ Right = b[6] – b[5] = 11 – 9 = 2
◦ **tongsocach = Left * Right = 2 * 2 = 4**

▪ Đoạn 4: [7, 9, 11] (tương tự như cách tính của Đoạn 2)

◦ Left = b[4] – b[3] = 7 – 6 = 1
◦ Right = b[7] – b[6] = 13 – 11 = 2
◦ **tongsocach = Left * Right = 1 * 2 = 2**

▪ Đoạn 1: [2, 4, 6] (tương tự như cách tính của Đoạn 2)

◦ b[0] = 0
◦ Left = b[1] – b[0] = 2 – 0 = 2
◦ Right = b[4] – b[3] = 7 – 6 = 1
◦ **tongsocach = Left * Right = 2 * 1 = 2**

▪ Đoạn 5: [9, 11, 13] (tương tự như cách tính của Đoạn 2)

◦ b[8] = N + 1 = 13 + 1 = 14
◦ Left = b[5] – b[4] = 9 – 7 = 2
◦ Right = b[8] – b[7] = 14 – 13 = 1

◦ **tongsocach = Left * Right = 2 * 1 = 2**

+ Cộng dồn từng tongsocach của mỗi đoạn ta sẽ có được kết quả theo yêu cầu của đề bài: **ketqua = 4 + 4 + 2 + 2 + 2 = 14**

- **Dữ liệu ra:** In ra ketqua

3.3. Mô tả thuật toán:

- **Bước 1. Đọc dữ liệu từ file**

+ Đọc 2 số nguyên **N, K** (số phần tử trong mảng **a** và số lượng số lẻ cần tìm).

+ Đọc mảng **a** gồm **N** phần tử.

- **Bước 2. Chuẩn hóa mảng a**

Chèn thêm phần tử **0** vào đầu mảng **a**.

=> Mục đích: để các chỉ số trong mảng bắt đầu từ **1** thay vì **0**, dễ thao tác.

- **Bước 3. Tìm vị trí các số lẻ xuất hiện trong mảng a rồi gán các vị trí này vào mảng b**

+ Tạo mảng rỗng **b**.

+ Duyệt từng phần tử **a[i] (i = 1 đến len(a))**:

▪ Nếu **a[i]** là số lẻ, thêm chỉ số **i** vào **b**.

+ Sau bước này, **b** lưu lại các vị trí xuất hiện của số lẻ trong mảng **a**.

- **Bước 4. Xác định số lượng đoạn con chứa đúng K phần tử lẻ trong mảng b**

+ Gọi **m** là số lượng số lẻ trong mảng **a**. **(m = len(b))**

+ Số đoạn con chứa đúng K số lẻ chính là đoạn con có độ dài **K** trong mảng **b**.

+ Công thức: **soluongdoanconbangK = m - K + 1**.

- **Bước 5. Thêm biên cho mảng b**

+ Thêm số **0** vào đầu mảng **b**.

+ Thêm số **N+1** vào cuối mảng **b**.

=> Đây là 2 "cận biên" giúp tính toán số đoạn con hợp lệ dễ dàng hơn.

- **Bước 6. Duyệt từng soluongdoanconbangK trong mảng b để tính kết quả**

+ Với mỗi **i** từ **1** đến **soluongdoanconbangK**:

▪ Xét từng đoạn con gồm **K** số lẻ liên tiếp: **[b[i], b[i+1], ..., b[(i+K)-1]]**.

▪ Tính:

◦ **left = b[i] - b[i-1]** (số cách lựa chọn từ vị trí bắt đầu của đoạn con đang xét trở về trước).

◦ **right = b[i + K] - b[(i + K) - 1]** (số cách lựa chọn bắt đầu từ vị trí kết thúc đoạn con đang xét trở về sau).

◦ Số đoạn con có đúng **K** số lẻ trong đoạn này sẽ là: **left * right**.

◦ **Cộng dồn** số đoạn con có đúng **K** số lẻ trong từng đoạn để tìm kết quả cuối cùng theo yêu cầu của đề bài và gán kết quả vào biến **ketqua**.

- **Bước 7. Xuất kết quả ra file**

Ghi kết quả cuối cùng **(ketqua)** vào file **output**

3.4. Code tham khảo (Python và C++):

3.4.1. Code Python:

```python
# Mo file de doc
fi = open('CHUMDEN.INP' , 'r')
# Nhap gia tri N, K
N, K = fi.readline().split()
N, K = int(N), int(K)
# Nhap mang a
a = fi.read().split()
a = list(map(int, a))
# Dong file
fi.close()
# Chen them so 0 vao dau mang a de co the bat dau tai vi tri 1 thay vi bat dau tai vi
tri 0 nhu mac dinh cho de hieu
a.insert(0, 0)
# Khoi tao mang b de luu vi tri xuat hien cua cac so le trong mang a
b = []
# Dung vong lap for de duyet mang a, tim vi tri cac so le roi gan vao mang b
for i in range(1, len(a)):
    if a[i] % 2 == 1: b.append(i)
# So luong so le m trong mang a
m = len(b)
# So luong doan con chua dung K phan tu le trong mang b
soluongdoanconbangK = m - K + 1
# Chen them so 0 vao dau mang b
b.insert(0, 0)
# Chen them so N + 1 vao sau mang b
b. append(N + 1)
# Tinh ketqua cua de bai theo cong thuc left, right
ketqua = 0
for i in range(1, soluongdoanconbangK + 1):
    left = b[i] - b[i - 1]
    right = b[i + K] - b[(i + K) - 1]
    ketqua = ketqua + left * right
# Mo file de ghi
fo = open('CHUMDEN.OUT', 'w')
# Ghi ket qua ra file
fo.write(f'{ketqua}')
# Dong file
fo.close()
```

3.4.2. Code C++:

```cpp
#include <iostream>
#include <bits/stdc++.h>
#include <fstream> // Khai bao cai nay de su dung ifstream va ofstream
#include <vector> // Khai bao cai nay de su dung vector
using namespace std;
int main()
```

```
{
    // Mo file de doc
    ifstream fi("CHUMDEN.INP");
    // Nhap gia tri N, K
    int N, K;
    fi >> N >> K;
    // Nhap mang a
    vector <int> a(N);
    for (int i = 0; i < N; i++) fi >> a[i];
    // Dong file
    fi.close();
    // Chen them so 0 vao dau mang a de co the bat dau tai vi tri 1 thay vi bat dau
tai vi tri 0 nhu mac dinh cho de hieu
    a.insert(a.begin(), 0);
    // Khoi tao mang b de luu vi tri xuat hien cua cac so le trong mang a
    vector <int> b;
    // Dung vong lap for de duyet mang a, tim vi tri cac so le roi gan vao mang b
    for (int i = 1; i < a.size(); i++)
    {
        if (a[i] % 2 == 1) b.push_back(i);
    }
    // So luong so le m trong mang a
    int m = b.size();
    // So luong doan con chua dung K phan tu le trong mang b
    int soluongdoanconbangK = m - K + 1;
    // Chen them so 0 vao dau mang b
    b.insert(b.begin(), 0);
    // Chen them so N + 1 vao sau mang b
    b.push_back(N + 1);
    // Tinh ketqua cua de bai theo cong thuc left, right
    long long ketqua = 0;
    for (int i = 1; i < soluongdoanconbangK + 1; i++)
    {
        int left  = b[i] - b[i - 1];
        int right = b[i + K] - b[(i + K) - 1];
        ketqua = ketqua + 1LL * left * right;
    }
    // Mo file de ghi
    ofstream fo("CHUMDEN.OUT");
    // Ghi ket qua ra file
    fo << ketqua;
    // Dong file
    fo.close();
    return 0;
}
```

4. Câu 4. TẶNG QUÀ

Câu 4. TẶNG QUÀ

Bố Tí là một người rất giàu có, ông có rất nhiều đất đai và các món đồ quý hiếm. Đặc biệt ông có bộ sưu tập gồm n món đồ cổ được đánh số thứ tự từ 1 đến n có giá trị cao. Ông đã nhờ các chuyên gia về đồ cổ định giá cho từng món đồ cổ của mình. Sau khi định giá các chuyên gia đã đưa ra giá trị của món đồ cổ thứ i là $a_i (\forall i = 1..n)$. Tí là đứa con duy nhất nên ông đã quyết định tặng cho Tí một số món từ bộ sưu tập đồ cổ của mình để làm vốn riêng. Ông cho Tí được tự ý lựa chọn các món đồ tuy nhiên có một yêu cầu cho Tí là các món chọn sau phải có số thứ tự và giá trị cao hơn món chọn trước đó.

Yêu cầu: Hãy giúp Tí tính xem phải chọn những món đồ trong bộ sưu tập đồ cổ như thế nào để số món đồ không được chọn là ít nhất.

Dữ liệu vào: Đọc từ tệp văn bản TANGQUA.INP

- Dòng thứ nhất chứa số nguyên dương $n (n \le 10^5)$ là số lượng các món đồ cổ.
- Dòng thứ hai ghi n số nguyên $a_1, a_2, \dots, a_n (1 \le a_i \le 10^9)$ là giá trị của từng món đồ.

Dữ liệu ra: Ghi ra tệp văn bản TANGQUA.OUT một số nguyên duy nhất là số món đồ cổ Tí không chọn.

Ví dụ:

TANGQUA.INP	TANGQUA.OUT	Giải thích
5 1 3 3 2 8	2	Tí chọn được 3 món đồ trong 5 món ở các vị trí lần lượt là (1, 3, 5)

Ràng buộc:

- Có 40% số test đầu với $n \le 25$;
- Có 30% số test tiếp theo với $25 < n \le 2000$;
- Có 30% số test còn lại không có ràng buộc gì thêm.

4.1. Tóm tắt đề bài:

Cho một mảng a gồm n phần tử. Hãy cho biết số lượng phần tử của dãy con không liến tiếp tăng dài nhất trong mảng a. (Tìm độ dài của dãy con không liên tiếp tăng dài nhất (LIS))

4.2. Ý tưởng giải:

- **Bước 1.** Khởi tạo mảng **L** chỉ chứa **n** các số có giá trị bằng **1**.
- **Bước 2.** Với mỗi **a[i]**, so sánh với các phần tử trước nó **a[j]**. Nếu **a[i] > a[j]** thì: **L[i] = max(L[i], L[j] + 1)**
- **Bước 3.** Kết quả = **max(L)**.

4.3. Mô tả thuật toán:

- **Bước 1.** Đọc dữ liệu từ file **TANGQUA.INP**
- **Bước 2.** Khởi tạo mảng **L** chỉ chứa **n** các số có giá trị bằng **1**

  L = [1] * n

  => L = [1, 1, …, 1] (**n** phần tử có giá trị = **1**)
- **Bước 2.** Sử dụng **2** vòng lặp **for** lồng nhau để duyệt:

  + Vòng lặp **for** thứ **1**: Cho **i** chạy từ **0** đến **n**

  + Vòng lặp **for** thứ **2**: Cho **j** chạy từ **0** đến **i**

  + **Nếu** phần tử **i** đang xét **(a[i])** mà lớn hơn các phần tử đứng trước nó **(a[j])** thì thay đổi L[i] thành max(L[i], L[j] + 1) **(L[i] = max(L[i], L[j] + 1)** => Đây chính là công thức quy hoạch động để tính L[i]).

  => Giải thích **Bước 2** bằng **Code**:

```
for i in range(0, n):
    for j in range(0, i):
        if a[i] > a[j]: L[i] = max(L[i], L[j] + 1)
```

- **Bước 3.** Kết quả cần tìm chính là giá trị lớn nhất của mảng **L (max(L))**
  ketqua = max(L) **(max(L)** chính là độ dài của dãy con không liên tiếp tăng dài nhất)
- **Bước 4.** Ghi kết quả ra file **TANGQUA.OUT**

4.4. Code tham khảo (Python và C++):

4.4.1. Code Python:

```python
# Mo file TANGQUA.INP de doc
fi = open("TANGQUA.INP", "r")
# Nhap gia tri n
n = int(fi.readline().strip())
# Nhap mang a
a = list(map(int, fi.readline().split()))
# Dong file sau khi doc xong
fi.close()
# Khoi tao mang L chi chua n so 1
L = [1] * n
# Su dung 2 vong lap for long nhau de duyet
for i in range(0, n):
    for j in range(0, i):
        if a[i] > a[j]: L[i] = max(L[i], L[j] + 1)
# Tinh ket qua cua de bai va luu vao bien ketqua
# Ket qua chinh la gia tri lon nhat cua mang L
ketqua = max(L)
# Mo file TANGQUA.OUT de ghi ket qua cua de bai
fo = open("TANGQUA.OUT", "w")
# Ghi ket qua ra file
fo.write(f'{ketqua}')
# Dong file TANGQUA.OUT sau khi da ghi xong ket qua
fo.close()
```

4.4.2. Code C++:

```cpp
#include <iostream>
#include <bits/stdc++.h>
#include <fstream> // Khai bao cai nay de su dung ifstream va ofstream
#include <vector> // Khai bao cai nay de su dung vector
using namespace std;
int main()
{
    // Mo file TANGQUA.INP de doc
    ifstream fi("TANGQUA.INP");
    // Nhap gia tri n
    int n;
    fi >> n;
    // Nhap mang a
    vector <int> a(n);
    for (int i = 0; i < n; i++) fi >> a[i];
    // Dong file sau khi doc xong
    fi.close();
```

```
    // Khoi tao mang L chi chua n so 1
    vector <int> L(n, 1);
    // Su dung 2 vong lap for long nhau de duyet
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < i; j++)
        {
            if (a[i] > a[j]) L[i] = max(L[i], L[j] +
1);
        }
    }
    // Tinh ket qua cua de bai va luu vao bien ketqua
    // Tim gia tri lon nhat cua mang L
    int max = L[0];
    for (int i = 1; i < n; i++)
    {
        if (L[i] > max) max = L[i];
    }
    // Tinh ket qua cua de bai va luu vao bien ketqua
    // Ket qua chinh la gia tri lon nhat cua mang L
    int ketqua = max;
    // Mo file TANGQUA.OUT de ghi ket qua cua de bai
    ofstream fo("TANGQUA.OUT");
    // Ghi ket qua ra file
    fo << ketqua;
    // Dong file TANGQUA.OUT sau khi da ghi xong ket qua
    fo.close();
    return 0;
}
```

5. Các bộ Test:

Mỗi câu của đề thi đều có đủ 10 bộ Test, với các giá trị đầu vào tương ứng với tỉ lệ (%) của các ràng buộc (Substack) mà đề bài yêu cầu. Những bộ Test này đã được gửi ở tệp tin đính kèm cùng với tài liệu này.

6. Kết quả chấm thử bằng phần mềm Themis 1.9, giới hạn thời gian chạy của mỗi bộ Test là 01 giây:

Tên bài	Điểm tối đa	Điểm chấm thử bằng Themis 1.9	
		Python	C++
Câu 1. TÍNH TỔNG (TONG.*)	3.0 điểm	3.0 điểm (100%)	3.0 điểm (100%)
Câu 2. OẰN TÙ XÌ (GAME.*)	3.0 điểm	3.0 điểm (100%)	3.0 điểm (100%)
Câu 3. CHÙM ĐÈN (CHUMDEN.*)	2.0 điểm	2.0 điểm (100%)	2.0 điểm (100%)
Câu 4. TẶNG QUÀ (TẶNG QUÀ.*)	2.0 điểm	1.4 điểm (70%)	1.8 điểm (90%)
Tổng số điểm đạt được:		**9.4 điểm (94%)**	**9.8 điểm (98%)**

ĐỀ SỐ 3
(Học sinh giỏi 9 – Đà Nẵng – Năm học: 2022 – 2023)

ĐỀ THI HỌC SINH GIỎI LỚP 9 MÔN TIN HỌC
THÀNH PHỐ ĐÀ NẴNG
NĂM HỌC 2022 – 2023
Thời gian làm bài: ... phút, không kể thời gian phát đề

TỔNG QUAN ĐỀ THI

Bài	Tên bài	Tên file CT	File input	File output	Điểm
1	Ước thực sự	UTS.*	UTS.INP	UTS.OUT	
2	Đếm ký tự	DKT.*	DKT.INP	DKT.OUT	
3	Số nguyên tố đối xứng	NTDX.*	NTDX.INP	NTDX.OUT	
4	Tổng các hoán vị	TONGHV.*	TONGHV.INP	TONGHV.OUT	

*Dấu * được thay thế bởi PAS, CPP hoặc PY tương ứng với ngôn ngữ lập trình sử dụng*

LẬP TRÌNH GIẢI CÁC BÀI TOÁN SAU

Bài 1. UTS Ước thực sự lớn nhất

Số tự nhiên y được gọi là một ước thực sự của số tự nhiên x nếu x chia hết cho y và x > y .

Yêu cầu: Nhập vào từ bàn phím một số nguyên dương x (với x > 1), hãy tìm và in ra màn hình số y là ước thực sự lớn nhất của số x.

Input: Một số nguyên dương x

Output: Ghi ra số nguyên y thỏa mãn yêu cầu của đề bài.

Giới hạn:

- Có 70% test tương ứng với $x \le 10^6$.
- Có 20% test tương ứng với $x \le 10^8$.
- Có 10% test tương ứng với $x \le 10^{10}$.

Ví dụ:

Input	Output	Giải thích
10	5	

Bài 2. DKT Đếm kí tự

Nhập từ bàn phím một xâu kí tự s.

Yêu cầu: Hãy đếm và in ra màn hình số lượng kí tự xuất hiện ít nhất 2 lần trong xâu kí tự s (có phân biệt chữ hoa và chữ thường).

Input: Xâu S có không quá 255 kí tự.

Output: Số lượng kí tự xuất hiện ít nhất 2 lần.

Ví dụ:

Input	Output	Giải thích
abcbMbdmcccd	3	

Bài 3. NTDX Số nguyên tố đối xứng

Một số được gọi là số nguyên tố đối xứng nếu nó là một số nguyên tố và khi đọc số này từ trái sang phải hay từ phải sang trái đều như nhau. Chẳng hạn số **131** là số nguyên tố đối xứng.

Yêu cầu: Cho trước một số nguyên dương x. Hãy tính xem có bao nhiêu số nguyên tố đối xứng lớn hơn 10 và bé hơn x.

Input: một số nguyên dương x

Output: Ghi số nguyên m thỏa mãn yêu cầu của đề bài.

1

Giới hạn:

- Có 50% test tương ứng với $x \le 10^4$.
- Có 30% test tương ứng với $x \le 10^6$.
- Có 20% test tương ứng với $x \le 10^{10}$.

Ví dụ:

Input	Output	Giải thích
150	3	Có 3 số nguyên tố đối xứng lớn hơn 10 và bé hơn 150 là 131; 101 và 11 .

Bài 4. TONGHV Tổng của các hoán vị

Cho trước một số tự nhiên x có n chữ số và không có 2 chữ số nào giống nhau. Người ta thay đổi trật tự các chữ số của số x cho nhau để được số tự nhiên mới cũng có n chữ số và mỗi số mới này được gọi là một hoán vị của số x.

Yêu cầu: Tính tổng của số x và tất cả các hoán vị của x.

Input: một số nguyên dương x

Output: Ghi số nguyên m thỏa mãn yêu cầu của đề bài.

Giới hạn:

- Có 30% test tương ứng với $x \le 10^3$.
- Có 20% test tương ứng với $x \le 10^5$.
- Có 30% test tương ứng với $x \le 10^8$.
- Có 20% test tương ứng với $x \le 10^{10}$.

Ví dụ:

Input	Output	Giải thích
123	1332	Tổng của số 123 và các hoán vị của nó là: 123 + 132 + 213 + 231 + 312 + 321 = 1332

BÀI GIẢI ĐỀ SỐ 3
(Học sinh giỏi 9 – Đà Nẵng – Năm học: 2022 – 2023)

1. Bài 1. UTS Ước thực sự lớn nhất

Bài 1. UTS Ước thực sự lớn nhất

Số tự nhiên y được gọi là một ước thực sự của số tự nhiên x nếu x chia hết cho y và x > y .

Yêu cầu: Nhập vào từ bàn phím một số nguyên dương x (với x > 1), hãy tìm và in ra màn hình số y là ước thực sự lớn nhất của số x.

Input: Một số nguyên dương x

Output: Ghi ra số nguyên y thỏa mãn yêu cầu của đề bài.

Giới hạn:

- Có 70% test tương ứng với $x \le 10^6$.
- Có 20% test tương ứng với $x \le 10^8$.
- Có 10% test tương ứng với $x \le 10^{10}$.

Ví dụ:

Input	Output	Giải thích
10	5	

1.1. Tóm tắt đề bài:

Nhập vào một số nguyên dương **x** ($1 < x <= 10^{10}$). In ra màn hình một số **y** là ước lớn nhất của **x** (**y** ≠ **x**).

1.2. Ý tưởng giải:

- Duyệt i từ 2 đến căn bậc hai của x:

  + Khi gặp số i đầu tiên chia hết cho x (tức là ước nhỏ nhất), thì ước lớn nhất tương ứng là y = x // i.

  + Nếu không tìm được ước nào, thì x là số nguyên tố → ước lớn nhất là y=1.

1.3. Mô tả thuật toán:

- **Bước 1.** Nạp thư viện **math** để dùng hàm **sqrt()** (căn bậc hai)
- **Bước 2.** Mở file **UTS.INP** để đọc dữ liệu vào
- **Bước 3. Tìm ước thực sự lớn nhất của x**

  + Duyệt **i** từ **2** đến **căn bậc hai của x**:

    Khi gặp số **i** đầu tiên chia hết cho **x** (tức là ước nhỏ nhất), thì đưa ra ước lớn nhất tương ứng là y = x // i và dừng vòng lặp ngay.

  + Nếu không tìm được ước nào, thì x là số nguyên tố → ước lớn nhất là 1 (y = 1).

- **Bước 4.** Ghi kết quả ra file **UTS.OUT**

1.4. Code tham khảo (Python và C++):

1.4.1. Code Python:

```python
# Bước 1. Nạp thư viện math để dùng hàm sqrt() (căn bậc hai)
import math
# Bước 2. Mở file UTS.INP để đọc dữ liệu vào
fi = open('UTS.INP', 'r')
x = int(fi.read())      # Đọc dữ liệu trong file và đổi sang kiểu số nguyên
fi.close()              # Đóng file sau khi đọc xong
# Bước 3. Tìm ước thực sự lớn nhất của x
for i in range(2, int(math.sqrt(x)) + 1):
    if x % i == 0:  # Nếu i chia hết x
        y = x // i  # Thì tìm được ước lớn nhất tương ứng
        break       # Dừng vòng lặp ngay, không cần kiểm tra thêm
else: y = 1         # Nếu vòng lặp kết thúc mà không có break (tức là không có
ước nào) => x là số nguyên tố → ước thực sự lớn nhất là y = 1
# Bước 4. Ghi kết quả ra file UTS.OUT
fo = open('UTS.OUT', 'w')   # Mở file để ghi
fo.write(f'{y}')            # Ghi giá trị y ra file (kết quả bài toán)
fo.close()                  # Đóng file sau khi ghi xong
```

1.4.2. Code C++:

```cpp
// Buoc 1. Nap thu vien cmath de dung ham sqrt() (can bac hai)
#include <iostream>
#include <bits/stdc++.h>
#include <fstream>  // Khai bao cai nay de su dung ifstream va ofstream
#include <cmath>    // Nap thu vien cmath de dung ham sqrt() (can bac hai)
using namespace std;
int main()
{
    // Buoc 2. Mo file UTS.INP de doc du lieu vao
    ifstream fi("UTS.INP");          // Mo file de doc
```

```
    long long x;                        // Khai bao bien x kieu so nguyen lon
    fi >> x;                            // Doc du lieu trong file va gan cho x
    fi.close();                         // Dong file sau khi doc xong
    // Buoc 3. Tim uoc thuc su lon nhat cua x
    long long y = 1;                    // Bien y luu uoc thuc su lon nhat ban
dau
    for (long long i = 2; i <= sqrt(x); i++)
    {
        if (x % i == 0)       // Neu i chia het x
        {
            y = x / i;        // Thi tim duoc uoc lon nhat tuong ung
            break;            // Dung vong lap ngay, khong can kiem tra them
        }
    }
    // Neu vong lap ket thuc ma khong co break (tuc la khong co uoc nao) => x la so
nguyen to → uoc thuc su lon nhat la y = 1 nhu ban dau da gan
    // Buoc 4. Ghi ket qua ra file UTS.OUT
    ofstream fo("UTS.OUT"); // Mo file de ghi
    fo << y;                // Ghi gia tri y ra file (ket qua bai toan)
    fo.close();             // Dong file sau khi ghi xong
    return 0;
}
```

2. Bài 2. DKT Đếm kí tự

Bài 2. DKT Đếm kí tự

Nhập từ bàn phím một xâu kí tự s.

Yêu cầu: Hãy đếm và in ra màn hình số lượng kí tự xuất hiện ít nhất 2 lần trong xâu kí tự s (có phân biệt chữ hoa và chữ thường).

Input: Xâu S có không quá 255 kí tự.

Output: Số lượng kí tự xuất hiện ít nhất 2 lần.

Ví dụ:

Input	Output	Giải thích
abcbMbdmccccd	3	

2.1. Tóm tắt đề bài:

- Nhập vào một xâu ký tự **s**.
- Đếm bao nhiêu ký tự xuất hiện ít nhất 2 lần (phân biệt chữ hoa và chữ thường).
- In ra kết quả.

2.2. Ý tưởng giải:

- Tạo một từ điển (dictionary) **dem** để đếm số lần xuất hiện của từng ký tự. Đầu tiên gán **dem{}** (**dem = rỗng**)
- Duyệt từng kí tự **kt** trong xâu **s**. Nếu kí tự đã xuất hiện thì tăng số lần **dem[kt]** lên **1**, nếu là lần đầu xuất hiện thì gán **dem[kt] = 1**
- Duyệt qua tất cả các ký tự trong từ điển **dem**. Nếu ký tự đó xuất hiện từ **2** lần trở lên thì tăng số lượng (**soluong**)
- In ra số lượng (**soluong**) cuối cùng

2.3. Mô tả thuật toán:

- **Bước 1.** Nhập xâu ký tự **s** từ file **DKT.INP**
- **Bước 2.** Tạo một từ điển (dictionary) **dem** để đếm số lần xuất hiện của từng ký tự

  + Ban đầu **dem = {}** (dem = rỗng)

  + Duyệt từng kí tự **kt** trong xâu **s**. Nếu kí tự đã xuất hiện thì tăng số lần **dem[kt]** lên **1**, nếu là lần đầu xuất hiện thì gán **dem[kt] = 1**
- **Bước 3.** Đếm số ký tự xuất hiện ít nhất 2 lần

  Duyệt qua tất cả các ký tự trong từ điển **dem**. Nếu ký tự đó xuất hiện từ **2** lần trở lên thì tăng số lượng (**soluong**) lên **1**
- **Bước 4.** In kết quả ra file **DKT.OUT**

2.4. Code tham khảo (Python và C++):

2.4.1. Code Python:

```python
# Bước 1. Nhập xâu ký tự s từ file DKT.INP
fi = open('DKT.INP', 'r')    # Mở file để đọc
s = fi.read()                # Nhập s
fi.close()                   # Đóng file sau khi đọc xong
# Bước 2. Tạo một từ điển (dictionary) dem để đếm số lần xuất hiện của từng ký tự
dem = {}                             # Ban đầu rỗng
for kt in s:                         # Duyệt từng ký tự trong xâu
    if kt in dem:                    # Nếu ký tự đã xuất hiện
        dem[kt] = dem[kt] + 1        # Tăng số lần đếm lên 1
    else:
        dem[kt] = 1                  # Nếu lần đầu xuất hiện thì gán = 1
# Bước 3. Đếm số lượng ký tự xuất hiện ít nhất 2 lần
soluong = 0
for kt in dem:                       # Duyệt qua tất cả các ký tự trong từ điển
(dictionary)
    if dem[kt] >= 2:                 # Nếu ký tự đó xuất hiện từ 2 lần trở lên
        soluong = soluong + 1        # Tăng số lượng
# Bước 4. In kết quả ra file DKT.OUT
fo = open('DKT.OUT', 'w')    # Mở file để ghi
fo.write(f'{soluong}')       # Ghi soluong ra file DKT.OUT
fo.close()                   # Đóng file sau khi ghi
```

2.4.2. Code C++:

```cpp
#include <iostream>
#include <bits/stdc++.h>
#include <fstream>       // Khai bao cai nay de su dung ifstream va ofstream
#include <map>           // Thu vien dung kieu du lieu map (tuong tu dictionary
trong Python)
using namespace std;
int main()
{
    // Buoc 1. Nhap xau ky tu s tu file DKT.INP
    ifstream fi("DKT.INP");    // Mo file de doc
    string s;
```

```
    getline(fi, s);             // Doc xau ky tu s trong file
    fi.close();                 // Dong file sau khi doc xong
    // Buoc 2. Tao mot tu dien (map) dem de dem so lan xuat hien cua tung ky tu
    map <char, int> dem;                // Ban dau rong
    for (char kt : s)                   // Duyet tung ky tu trong xau
    {
        if (dem.count(kt))              // Neu ky tu da xuat hien
            dem[kt] = dem[kt] + 1;      // Tang so lan dem len 1
        else
            dem[kt] = 1;                // Neu lan dau xuat hien thi gan = 1
    }
    // Buoc 3. Dem so luong ky tu xuat hien it nhat 2 lan
    int soluong = 0;
    for (auto kt : dem)                 // Duyet qua tat ca cac ky tu trong map
    {
        if (kt.second >= 2)             // Neu ky tu do xuat hien tu 2 lan tro
len
            soluong = soluong + 1;      // Tang so luong
    }
    // Buoc 4. In ket qua ra file DKT.OUT
    ofstream fo("DKT.OUT"); // Mo file de ghi
    fo << soluong;          // Ghi gia tri ra file DKT.OUT
    fo.close();             // Dong file sau khi ghi xong
    return 0;
}
```

3. Bài 3. NTDX Số nguyên tố đối xứng

Bài 3. NTDX Số nguyên tố đối xứng

Một số được gọi là số nguyên tố đối xứng nếu nó là một số nguyên tố và khi đọc số này từ trái sang phải hay từ phải sang trái đều như nhau. Chẳng hạn số **131** là số nguyên tố đối xứng.

Yêu cầu: Cho trước một số nguyên dương x. Hãy tính xem có bao nhiêu số nguyên tố đối xứng lớn hơn 10 và bé hơn x.

Input: một số nguyên dương x

Output: Ghi số nguyên m thỏa mãn yêu cầu của đề bài.

Giới hạn:

- Có 50% test tương ứng với $x \le 10^4$.
- Có 30% test tương ứng với $x \le 10^6$.
- Có 20% test tương ứng với $x \le 10^{10}$.

Ví dụ:

Input	Output	Giải thích
150	3	Có 3 số nguyên tố đối xứng lớn hơn 10 và bé hơn 150 là 131; 101 và 11 .

3.1. Tóm tắt đề bài:

Cho trước một số nguyên dương x. Hãy tính xem có bao nhiêu số nguyên tố đối xứng lớn hơn 10 và bé hơn x.

3.2. Ý tưởng giải:

- Khai báo hàm kiểm tra số nguyên tố
- Nhập **x**
- Sinh ra tất cả các **số đối xứng** nhỏ hơn **x**
- Kiểm tra xem **số đối xứng** nào là số nguyên tố
- Đếm số lượng số đối xứng là số nguyên tố và in ra kết quả

3.3. Mô tả thuật toán:

- **Bước 1.** Khai báo hàm kiểm tra số nguyên tố.
- **Bước 2.** Nhập số nguyên dương **x** từ file.
- **Bước 3.** Đặt biến **dem = 0** để đếm số lượng nguyên tố đối xứng tìm được.
- **Bước 4.** Duyệt qua các độ dài nửa đầu của số đối xứng có thể sinh ra (**nua_dau**).
- **Bước 5.** Với mỗi **nua_dau**, tạo ra:
  + Số đối xứng chẵn → ghép **nua_dau + phần đảo ngược**.
  + Số đối xứng lẻ → ghép **nua_dau + phần đảo ngược (bỏ chữ cuối)**.
  + Ví dụ:

Nửa đầu (nua_dau)	Đảo (dao)	Số đối xứng chẵn	Đảo (bỏ chữ cuối)	Số đối xứng lẻ
123	321	123321	21	12321

- **Bước 6.** Với mỗi số sinh được:
  Nếu nhỏ hơn **x** và là số nguyên tố → tăng biến đếm.
- **Bước 7.** In ra kết quả cuối cùng.

3.4. Code tham khảo (Python và C++):

3.4.1. Code Python:

```
import math # Khai báo thư viện toán học math, thường dùng khi chương trình có tính
căn bậc 2
# Tạo hàm kiểm tra số nguyên tố
def nguyento(x):
    # Nếu x <= 1 thì không phải số nguyên tố
    if x <= 1: return False
    # Nếu x là 2 hoặc 3 thì là số nguyên tố
    if x == 2 or x == 3: return True
    # Nếu x chia hết cho 2 hoặc 3 thì không phải số nguyên tố
    if x % 2 == 0 or x % 3 == 0: return False
    # Bắt đầu từ 5 (vì số nguyên tố > 3 có dạng 6k ± 1)
    i = 5
    # Kiểm tra đến căn bậc hai của x
    while i <= math.isqrt(x):
        # Nếu chia hết cho i hoặc i+2 thì không phải số nguyên tố
        if x % i == 0 or x % (i + 2) == 0:
            return False
        # Mỗi bước tăng 6 (kiểm tra cặp 6k-1 và 6k+1)
        i = i + 6
    # Nếu không chia hết cho số nào, x là số nguyên tố
    return True
```

```
# ===== CHƯƠNG TRÌNH CHÍNH =====
fi = open('NTDX.INP', 'r')      # Mở file để doc
x = fi.read()                   # Nhập x
x = int(x)
fi.close()                      # Đóng file sau khi doc xong
dem = 0  # Biến đếm số lượng nguyên tố đối xứng
# Duyệt theo độ dài của nửa đầu (để sinh số đối xứng nhanh hơn)
for do_dai in range(1, len(str(x)) // 2 + 2):
    # Sinh tất cả các nửa đầu có đúng 'do_dai' chữ số
    bat_dau = 10**(do_dai - 1)
    ket_thuc = 10**do_dai
    for nua_dau in range(bat_dau, ket_thuc):
        s = str(nua_dau)                    # Chuyển nửa đầu thành chuỗi
        # ===== TẠO SỐ ĐỐI XỨNG CHẴN =====
        dao = ''.join(reversed(s))          # Đảo ngược chuỗi
        doi_xung_chan = int(s + dao)        # Ghép lại thành số đối xứng
chẵn
        # ===== TẠO SỐ ĐỐI XỨNG LẺ =====
        dao_le = ''.join(reversed(s[:-1]))  # Bỏ ký tự cuối rồi đảo
        doi_xung_le = int(s + dao_le)       # Ghép lại thành số đối xứng
lẻ
        # ===== KIỂM TRA NGUYÊN TỐ VÀ ĐẾM =====
        if 10 < doi_xung_chan < x and nguyento(doi_xung_chan): dem
= dem + 1
        if 10 < doi_xung_le < x and nguyento(doi_xung_le): dem =
dem + 1
# ===== IN KẾT QUẢ =====
fo = open('NTDX.OUT', 'w')  # Mở file để ghi
fo.write(f'{dem}')          # Ghi kết quả ra file
fo.close()                  # Đóng file sau khi ghi xong
```

3.4.2. Code C++:

```
#include <iostream>
#include <bits/stdc++.h>     // Thu vien tong hop
#include <fstream>           // Khai bao cai nay de su dung ifstream va ofstream
using namespace std;
// ===== HAM KIEM TRA NGUYEN TO =====
bool nguyento(long long x)
{
    // Neu x <= 1 thi khong phai so nguyen to
    if (x <= 1) return false;
    // Neu x la 2 hoac 3 thi la so nguyen to
    if (x == 2 || x == 3) return true;
    // Neu x chia het cho 2 hoac 3 thi khong phai so nguyen to
    if (x % 2 == 0 || x % 3 == 0) return false;
    // Bat dau kiem tra tu 5 (vi so nguyen to > 3 co dang 6k ± 1)
    long long i = 5;
```

```
    // Kiem tra den can bac hai cua x
    while (i <= sqrt(x))
    {
        // Neu x chia het cho i hoac i+2 thi khong phai so nguyen to
        if (x % i == 0 || x % (i + 2) == 0) return false;
        // Moi buoc tang 6 (kiem tra cap 6k-1 va 6k+1)
        i = i + 6;
    }
    // Neu khong chia het cho so nao, x la so nguyen to
    return true;
}
// ===== CHUONG TRINH CHINH =====
int main()
{
    ifstream fi("NTDX.INP");     // Mo file de doc
    long long x;
    fi >> x;                     // Nhap x
    fi.close();                  // Dong file sau khi doc xong
    long long dem = 0;           // Bien dem so luong nguyen to doi xung
    // Duyet theo do dai cua nua dau (de sinh so doi xung nhanh hon)
    for (int do_dai = 1; do_dai <= (to_string(x).length() / 2 + 1);
do_dai++)
    {
        // Sinh tat ca cac nua dau co dung “do_dai” chu so
        long long bat_dau = pow(10, do_dai - 1);
        long long ket_thuc = pow(10, do_dai);
        for (long long nua_dau = bat_dau; nua_dau < ket_thuc;
nua_dau++)
        {
            string s = to_string(nua_dau);               // Chuyen nua
dau thanh chuoi
            // ===== TAO SO DOI XUNG CHAN =====
            string dao = s;
            reverse(dao.begin(), dao.end());             // Dao nguoc
chuoi
            long long doi_xung_chan = stoll(s + dao);    // Ghep lai
thanh so doi xung chan
            // ===== TAO SO DOI XUNG LE =====
            string dao_le = s.substr(0, s.size() - 1);   // Bo ky tu
cuoi roi dao
            reverse(dao_le.begin(), dao_le.end());
            long long doi_xung_le = stoll(s + dao_le);   // Ghep lai
thanh so doi xung le
            // ===== KIEM TRA TINH NGUYEN TO VA DIEU KIEN =====
            if (doi_xung_chan < x && doi_xung_chan > 10 &&
nguyento(doi_xung_chan)) dem = dem + 1;
```

```
            if (doi_xung_le < x && doi_xung_le > 10 &&
nguyento(doi_xung_le)) dem = dem + 1;
        }
    }
    // ===== IN KET QUA =====
    ofstream fo("NTDX.OUT");      // Mo file ghi ket qua
    fo << dem;                    // Ghi ra so luong dem duoc
    fo.close();                   // Dong file sau khi ghi xong
    return 0;
}
```

4. Bài 4. TONGHV Tổng của các hoán vị

Bài 4. TONGHV Tổng của các hoán vị

Cho trước một số tự nhiên x có n chữ số và không có 2 chữ số nào giống nhau. Người ta thay đổi trật tự các chữ số của số x cho nhau để được số tự nhiên mới cũng có n chữ số và mỗi số mới này được gọi là một hoán vị của số x.

Yêu cầu: Tính tổng của số x và tất cả các hoán vị của x.

Input: một số nguyên dương x

Output: Ghi số nguyên m thỏa mãn yêu cầu của đề bài.

Giới hạn:

- Có 30% test tương ứng với $x \le 10^3$.
- Có 20% test tương ứng với $x \le 10^5$.
- Có 30% test tương ứng với $x \le 10^8$.
- Có 20% test tương ứng với $x \le 10^{10}$.

Ví dụ:

Input	Output	Giải thích
123	1332	Tổng của số 123 và các hoán vị của nó là: 123 + 132 + 213 + 231 + 312 + 321 = 1332

4.1. Tóm tắt đề bài:

- Nhập số nguyên **x**.
- Tính **tổng của x và tất cả các hoán vị của x**.
- In ra tổng đó.

4.2. Ý tưởng giải:

- Sử dụng công thức sau để tính tổng của tất cả các hoán vị của **x**:

$$S = (n-1)! \times \frac{10^n - 1}{9} \times (\text{tổng các chữ số})$$

- Trong đó:
  + **S**: là tổng của các hoán vị (kết quả cần tìm)
  + **n**: là số lượng chữ số của **x**
  + **tổng các chữ số**: là tổng các chữ số có trong **x** (Ví **dụ x = 624** => tổng các chữ số có trong **x** là **tong = 6 + 2 + 4 = 12**, số lượng chữ số của **x** là **n = 3**)

4.3. Mô tả thuật toán:

- **Bước 1.** Nhập **x** từ file **TONGHV.INP**
- **Bước 2.** Tính tổng các chữ số **(tong)** có trong **x** và tìm số lượng chữ số **(n)** có trong **x**
- **Bước 3.** Tính tích từ **1** đến **n – 1 (tich = 1 * 2 * 3 * ... * n - 1)**
- **Bước 4.** Tính kết quả **m**:

  m = tich * ((10n – 1) // 9) * tong**
- **Bước 5.** In kết quả **m** ra file **TONGHV.OUT**

4.4. Code tham khảo (Python và C++):

4.4.1. Code Python:

```python
fi = open('TONGHV.INP', 'r')    # Mở file TONGHV.INP để đọc
x = fi.read()                   # Đọc toàn bộ nội dung file vào biến x (dạng chuỗi)
x = int(x)                      # Chuyển chuỗi thành số nguyên (ví dụ '123' -> 123)
fi.close()                      # Đóng file sau khi đọc xong
n = 0                           # Khởi tạo n = 0 (dùng để đếm số chữ số)
tong = 0                        # Khởi tạo tong = 0 (dùng để lưu tổng các chữ số)
while x != 0:                   # Vòng lặp tách từng chữ số của x (bằng toán học)
    tong = tong + x % 10        # Lấy chữ số cuối (x % 10) và cộng vào tong
    n = n + 1                   # Tăng số lượng chữ số lên 1
    x = x // 10                 # Bỏ chữ số cuối đi (chia lấy phần nguyên)
tich = 1                        # Khởi tạo tich = 1 (dùng để tính (n-1)!)
for i in range(1, n):           # i chạy từ 1 đến n-1
    tich = tich * i             # tich = 1*1*2*...*(n-1) => tich = (n-1)!
m = tich * ((10**n - 1) // 9) * tong # Công thức tính tổng giá trị m của các hoán vị
fo = open('TONGHV.OUT', 'w')    # Mở file TONGHV.OUT để ghi kết quả
fo.write(f'{m}')                # Ghi kết quả m vào file
fo.close()                      # Đóng file sau khi ghi xong
```

4.4.2. Code C++:

```cpp
#include <iostream>
#include <bits/stdc++.h>          // Thu vien tong hop
#include <fstream>                // Khai bao cai nay de su dung ifstream va ofstream
using namespace std;
int main()
{
    ifstream fi("TONGHV.INP");   // Mo file TONGHV.INP de doc du lieu
    long long x;
```

```
    fi >> x;                        // Doc gia tri x tu file
    fi.close();                     // Dong file sau khi đọc xong
    int n = 0;                      // Khoi tao n = 0 (dung de dem so chu so)
    long long tong = 0;             // Khoi tao tong = 0 (dung de luu tong cac chu
so)
    while (x != 0)                  // Vong lap tach tung chu so cua x (bang toan
hoc)
    {
        tong = tong + x % 10;       // Lay chu so cuoi (x % 10) va cong vao tong
        n = n + 1;                  // Tang so luong chu so len 1
        x = x / 10;                 // Bo chu so cuoi di (chia lay phan nguyen)
    }
    long long tich = 1;             // Khoi tao tich = 1 (dung de tinh (n-1)!)
    for (int i = 1; i < n; i++)     // i chay tu 1 den n-1
        tich = tich * i;            // tich = 1*1*2*...*(n-1) => tich = (n-1)!
    // --- Tinh 10^n ---
    long long luythua = 1;
    for (int i = 0; i < n; i++) luythua = luythua * 10;
    long long m = tich * ((luythua - 1) / 9) * tong;     // Cong thuc
tinh tong gia tri m cua cac hoan vị
    ofstream fo("TONGHV.OUT");      // Mo file TONGHV.OUT de ghi ket qua
    fo << m;                        // Ghi ket qua m vao file
    fo.close();                     // Dong file sau khi ghi xong
    return 0;                       // Ket thuc chuong trinh
}
```

5. Các bộ Test:

Mỗi câu của đề thi đều có đủ 10 bộ Test, với các giá trị đầu vào tương ứng với tỉ lệ (%) của các ràng buộc (Substack) mà đề bài yêu cầu. Những bộ Test này đã được gửi ở tệp tin đính kèm cùng với tài liệu này.

6. Kết quả chấm thử bằng phần mềm Themis 1.9, giới hạn thời gian chạy của mỗi bộ Test là 01 giây:

Tên bài	Điểm tối đa	Điểm chấm thử bằng Themis 1.9	
		Python	C++
Bài 1. Ước thực sự (UTS.*)	2.5 điểm	2.5 điểm (100%)	2.5 điểm (100%)
Bài 2. Đếm kí tự (DKT.*)	2.5 điểm	2.5 điểm (100%)	2.5 điểm (100%)
Bài 3. Số nguyên tố đối xứng (NTDX.*)	2.5 điểm	2.0 điểm (80%)	2.0 điểm (80%)
Bài 4. Tổng các hoán vị (TONGHV.*)	2.5 điểm	2.5 điểm (100%)	2.5 điểm (100%)
Tổng số điểm đạt được:		**9.5 điểm (95%)**	**9.5 điểm (95%)**

ĐỀ SỐ 4
(Tin học trẻ Bảng B (THCS) – Đà Nẵng – Năm học: 2022 – 2023)

HỘI THI TIN HỌC TRẺ
THÀNH PHỐ ĐÀ NẴNG
NĂM HỌC 2022-2023

Thời gian: 120 phút (không kể thời gian giao đề)
(Đề thi có 02 trang)

ĐỀ CHÍNH THỨC

TỔNG QUAN

Bài	Tên bài	File chương trình	File dữ liệu vào	File dữ liệu ra
1	CHUẨN HÓA	CHUANHOA.*	Từ bàn phím	Màn hình
2	TRUNG BÌNH CỘNG	TBCONG.*	TBCONG.INP	TBCONG.OUT
3	BỘ BA SỐ	BOBASO.*	BOBASO.INP	BOBASO.OUT
4	RÚT TIỀN	TIEN.*	TIEN.INP	TIEN.OUT

*Chú ý: Dấu * có thể là pas; c; cpp hoặc py tương ứng với ngôn ngữ lập trình là Free Pacal; C/C++ hoặc Python.*

BÀI 1. CHUẨN HÓA *(3 điểm)*

Nhập vào một xâu s là họ tên của một người, hãy chuẩn hóa lại họ tên đó. Xâu họ tên là xâu chuẩn hóa khi thỏa mãn các điều sau:

- Không có dấu cách thừa.
- Chữ cái đầu mỗi từ viết in hoa còn lại là in thường.

Dữ liệu vào: Nhập từ bàn phím một xâu s là họ tên của một người có độ dài không quá 255 kí tự.

Kết quả: In ra màn hình xâu họ tên sau khi được chuẩn hóa.

Ví dụ:

Bàn phím	Màn hình
NgUyen vAn Ca	Nguyen Van Ca
lE vaN TEO	Le Van Teo

BÀI 2. TRUNG BÌNH CỘNG *(3 điểm)*

Cho số nguyên n và dãy gồm n số nguyên $a_1, a_2, \dots, a_n$. Tìm cách xóa đi tối đa 2 dãy con liên tiếp không cắt nhau của dãy đó để các phần tử còn lại có trung bình cộng lớn nhất.

Dữ liệu vào: Đọc từ file văn bản TBCONG.INP gồm:

- Dòng đầu tiên chứa số nguyên dương $n (1 \le n \le 10^6)$ là số phần tử của dãy số.
- Dòng thứ hai gồm n số nguyên $a_1, a_2, \dots, a_n (|a_i| \le 10^9)$ là các phần tử của dãy.

Kết quả: Ghi ra file văn bản TBCONG.OUT một số nguyên duy nhất là phần nguyên trung bình cộng của các phần tử còn lại.

Ví dụ:

TBCONG.INP	TBCONG.OUT	Giải thích
5 2 5 5 1 3	5	Xóa đi đi 2 dãy [1,1] và [4,5]

Ràng buộc:

- Có 30% số test với $n \le 10^2$;
- Có 30% số test tiếp theo với $n \le 10^3$;
- Có 40% số test còn lại với $n \le 10^6$.

THT bảng B NH 2022 - 2023 *Trang 1/2*

BÀI 3. BỘ BA SỐ *(2 điểm)*

Cho n số nguyên không âm $a_1, a_2, \dots, a_n$ và một số nguyên dương m. Hãy đếm số bộ ba số (i, j, k) mà $a_i * a_j * a_k$ chia hết cho m (lưu ý nếu 2 bộ ba mà bộ này là hoán vị của bộ kia thì vẫn tính là 2 bộ, ví dụ $(1, 2, 3)$ và $(2, 1, 3)$ là hai bộ khác nhau).

Dữ liệu vào: Đọc từ file văn bản BOBASO.INP gồm:

- Dòng đầu tiên chứa 2 số nguyên n và $m (1 \le n \le 2.10^3, 1 \le m \le 3.10^3)$.
- Dòng thứ hai chứa n số nguyên không âm $a_1, a_2, \dots, a_n (0 \le a_i \le 10^9)$.

Kết quả: Ghi ra file văn bản BOBASO.OUT một dòng là số bộ ba số thỏa mãn yêu cầu.

Ví dụ:

BOBASO.INP	BOBASO.OUT	Giải thích
2 5 1 5	7	Có 7 bộ ba là (1,1,5), (1,5,1), (1,5,5), (5,1,1), (5,1,5), (5,5,1), (5,5,5)
10 3 1 2 3 4 5 6 7 8 9 10	657	Có tất cả 657 bộ ba số thỏa mãn chia hết cho 3.

Ràng buộc:

- Có 50% số test với $1 \le n \le 200$;
- Có 50% số test còn lại với $200 < m \le 2.10^3$.

BÀI 4: RÚT TIỀN *(2 điểm)*

An có rất nhiều tiền trong ngân hàng Thụy Sĩ, một hôm An cần rút một số tiền $N (N \le 10^5)$, ngân hàng chỉ có $K (K \le 10^3)$ loại mệnh giá lần lượt là $A_1, A_2, \dots, A_K$. Vì lí do đặc biệt nên An mong muốn số tờ tiền rút được là ít nhất.

Dữ liệu vào: Đọc từ file văn bản TIEN.INP gồm:

- Dòng thứ nhất chứa 2 số nguyên dương N và K. Trong đó N là số tiền cần rút, K là loại tiền mệnh giá.
- Dòng thứ hai chứa K số nguyên dương $A_1, A_2, \dots, A_K$ lần lượt là mệnh giá của các tờ tiền.

Kết quả: Ghi ra file văn bản TIEN.OUT chứa 1 số nguyên dương duy nhất là số tờ tiền ít nhất mà An rút được, nếu không thể in ra -1

Ví dụ:

TIEN.INP	TIEN.OUT	Giải thích
125 6 1 2 5 10 20 50	4	Số tờ tiền ít nhất có thể lấy là 4 tờ gồm 2 tờ mệnh giá 50, 1 tờ mệnh giá 20, một tờ mệnh giá 5.
5 3 2 4 6	-1	Không có cách nào để từ các tờ tiền mệnh giá 2 4 6 tạo thành số tiền là 5 cho nên ta in ra -1.

Ràng buộc:

- Có 20% số test có $K = 2$ và A_i khác nhau từng đôi một;
- Có 30% số test tiếp theo có $K \le 10, N \le 100$ và A_i khác nhau từng đôi một;
- Có 50% số test còn lại không có giới hạn gì khác.

—— Hết ——

Họ và tên thí sinh: ..... Số báo danh: .....Phòng thi .....

BÀI GIẢI ĐỀ SỐ 4
(Tin học trẻ Bảng B (THCS) – Đà Nẵng – Năm học: 2022 – 2023)

1. BÀI 1. CHUẨN HÓA *(3 điểm)*

BÀI 1. CHUẨN HÓA *(3 điểm)*

Nhập vào một xâu *s* là họ tên của một người, hãy chuẩn hóa lại họ tên đó. Xâu họ tên là xâu chuẩn hóa khi thỏa mãn các điều sau:

- Không có dấu cách thừa.
- Chữ cái đầu mỗi từ viết in hoa còn lại là in thường.

Dữ liệu vào: Nhập từ bàn phím một xâu *s* là họ tên của một người có độ dài không quá 255 kí tự.

Kết quả: In ra màn hình xâu họ tên sau khi được chuẩn hóa.

Ví dụ:

Bàn phím	Màn hình
NgUyen vAn Ca	Nguyen Van Ca
lE vaN TEO	Le Van Teo

1.1. Tóm tắt đề bài:

- **Input:** Một xâu (chuỗi) **s** là Họ và tên đệm, tên của một người (có thể nhập lẫn lộn chữ hoa, chữ thường, dư thừa dấu cách).
- **Output:** Chuỗi họ tên đã được chuẩn hóa theo dạng chuẩn (mỗi từ viết hoa chữ cái đầu, các chữ cái còn lại viết thường, giữa các từ cách nhau đúng một khoảng trắng, không có khoảng trắng thừa cuối cùng).

1.2. Ý tưởng giải:

- **Bước 1.** Nhập vào xâu **s** là Họ và tên của một người
- **Bước 2.** Tách từng từ trong xâu **s** ra và gán từng từ vừa tách vào trong mảng **a**
- **Bước 3.** Khởi tạo xâu **chuanhoa** để lưu lại kết quả của đề bài
- **Bước 4.** Ghép các phần tử trong mảng **a** lại để tạo thành một xâu duy nhất, giữa các từ trong xâu chỉ chứa 1 khoảng trắng
- **Bước 5.** Chuyển kí tự đầu của mỗi từ trong xâu chuanhoa thành chữ hoa, tất cả các kí tự còn lại của từ thành chữ thường
- **Bước 6.** Loại bỏ khoảng trắng cuối cùng trong xâu vừa chuẩn hóa
- **Bước 7.** Ghi kết quả ra file

1.3. Mô tả thuật toán:

- **Bước 1.** Đọc dữ liệu từ file **CHUANHOA.INP**
- **Bước 2.** Tách từng từ trong xâu **s** ra và gán từng từ vừa tách vào trong mảng **a**

+ **Python:**

```
a = s.split()
```

=> **split()** sẽ tự động loại bỏ khoảng trắng thừa và chia thành danh sách các từ.

+ C++:

```
string tu;
stringstream ss(s);
vector <string> a;
while (ss >> tu) a.push_back(tu);
```

- **Bước 3.** Khởi tạo xâu **chuanhoa** để lưu lại kết quả của đề bài

```
chuanhoa = ''
```

- **Bước 4.** Ghép các phần tử trong mảng **a** lại để tạo thành một xâu duy nhất, giữa các từ trong xâu chỉ chứa 1 khoảng trắng:

```
for i in range(0, len(a)):
    chuanhoa = chuanhoa + a[i] + ' '
```

(Ở đây sau vòng lặp for sẽ dư 1 dấu cách cuối).

- **Bước 5.** Chuyển kí tự đầu của mỗi từ trong xâu **chuanhoa** thành chữ hoa, tất cả các kí tự còn lại của từ thành chữ thường

```
chuanhoa = chuanhoa.title()
```

- **Bước 6.** Loại bỏ khoảng trắng cuối cùng trong xâu vừa chuẩn hóa

```
chuanhoa = chuanhoa.rstrip()
```

- **Bước 6.** Ghi kết quả ra file **CHUANHOA.OUT**

1.4. Code tham khảo (Python và C++):

1.4.1. Code Python:

```
# Mo file CHUANHOA.INP de doc
fi = open('CHUANHOA.INP', 'r')
# Nhap vao xau s la Ho va ten dem cua mot nguoi
s = fi.read()
# Dong file
fi.close()
# Tach tung tu trong s ra va gan tung tu vua tach vao trong mang a
a = s.split()
# Khoi tao xau chuanhoa de luu lai ket qua cua de bai
chuanhoa = ''
# Duyet tu phan tu dau mang den phan tu cuoi cua mang a, roi Ghep cac phan tu trong mang
a lai de tao thanh mot xau duy nhat, giua cac tu trong xau chi chua 1 khoang trang
for i in range(0, len(a)): chuanhoa = chuanhoa + a[i] + ' '
# Chuyen ki tu dau cua moi tu thanh chu hoa, tat ca cac ki tu con lai cua tu thanh chu
thuong theo lenh sau
chuanhoa = chuanhoa.title()
# Loai bo khoang trang cuoi cung trong xau chuanhoa
chuanhoa = chuanhoa.rstrip()
# Mo file de ghi
fo = open('CHUANHOA.OUT', 'w')
# Ghi ket qua ra file
fo.write(chuanhoa)
# Dong file
fo.close()
```

1.4.2. Code C++:

```
#include <iostream>
#include <bits/stdc++.h>
#include <fstream> // Khai bao cai nay de su dung ifstream va ofstream
#include <vector> // Khai bao cai nay de su dung vector
using namespace std;
```

```
int main()
{
    // Mo file CHUANHOA.INP de doc
    ifstream fi("CHUANHOA.INP");
    // Nhap vao xau s la Ho va ten dem cua mot nguoi
    string s;
    getline(fi, s);
    // Dong file
    fi.close();
    // Tach tung tu trong s ra va gan tung tu vua tach vao trong mang a
    string tu;
    stringstream ss(s);
    vector <string> a;
    while (ss >> tu) a.push_back(tu);
    // Khoi tao xau chuanhoa de luu lai ket qua cua de bai
    string chuanhoa = "";
    // Duyet tu phan tu dau mang den phan tu cuoi cua mang a, roi Ghep cac phan tu trong
mang a lai de tao thanh mot xau duy nhat, giua cac tu trong xau chi chua 1 khoang trang
    for (int i = 0; i < a.size(); i++)
    {
        chuanhoa = chuanhoa + a[i] + " ";
    }
    // Chuyen ki tu dau cua moi tu thanh chu hoa, tat ca cac ki tu con lai cua tu thanh
chu thuong theo lenh sau
    for (int i = 0; i < chuanhoa.size(); i++)
    {
        if (i == 0 || chuanhoa[i-1] == ' ')
            chuanhoa[i] = toupper(chuanhoa[i]);
        else
            chuanhoa[i] = tolower(chuanhoa[i]);
    }
    // Loai bo khoang trang cuoi cung trong xau chuanhoa
      if (chuanhoa.size() > 0 && chuanhoa.back() == ' ')
chuanhoa.pop_back();
    // Mo file de ghi
    ofstream fo("CHUANHOA.OUT");
    // Ghi ket qua ra file
    fo << chuanhoa;
    // Dong file
    fo.close();
    return 0;
}
```

2. BÀI 2. TRUNG BÌNH CỘNG *(3 điểm)*

BÀI 2. TRUNG BÌNH CỘNG ***(3 điểm)***

Cho số nguyên n và dãy gồm n số nguyên $a_1, a_2, \dots, a_n$. Tìm cách xóa đi tối đa 2 dãy con liên tiếp không cắt nhau của dãy đó để các phần tử còn lại có trung bình cộng lớn nhất.

Dữ liệu vào: Đọc từ file văn bản TBCONG.INP gồm:

- Dòng đầu tiên chứa số nguyên dương $n(1 \le n \le 10^6)$ là số phần tử của dãy số.
- Dòng thứ hai gồm n số nguyên $a_1, a_2, \dots, a_n(|a_i| \le 10^9)$ là các phần tử của dãy.

Kết quả: Ghi ra file văn bản TBCONG.OUT một số nguyên duy nhất là phần nguyên trung bình cộng của các phần tử còn lại.

Ví dụ:

TBCONG.INP	TBCONG.OUT	Giải thích
5 2 5 5 1 3	5	Xóa đi đi 2 dãy [1,1] và [4,5]

Ràng buộc:

- Có 30% số test với $n \le 10^2$;
- Có 30% số test tiếp theo với $n \le 10^3$;
- Có 40% số test còn lại với $n \le 10^6$.

2.1. Tóm tắt đề bài:

- Cho dãy số nguyên a[1..n].
- Được phép xóa đi tối đa 2 đoạn con liên tiếp (không giao nhau).
- Sau khi xóa, trung bình cộng của các số còn lại phải lớn nhất.
- In ra phần nguyên của giá trị trung bình đó.

2.2. Ý tưởng giải:

- **Muốn trung bình cộng lớn nhất → phải xóa đi những đoạn có tổng "xấu" nhất:**

  + Nếu ta không xóa: trung bình = tổng / n.
  + Nếu xóa 1 đoạn [L..R]: trung bình = (tổng – tổng[L..R]) / (n – (R-L+1)).
  + Nếu xóa 2 đoạn rời nhau:

    trung bình = (tổng – tổng_2_đoạn) / (n – độ_dài_2_đoạn).

  => Vậy, để trung bình lớn nhất, ta nên bỏ những đoạn con có **tổng nhỏ nhất**.

- **Chia thành 3 trường hợp cần xét:**

  + Không xóa gì → lấy trung bình gốc.

  + Xóa 1 đoạn liên tiếp → tìm đoạn có tổng nhỏ nhất, xóa đi.

  + Xóa 2 đoạn liên tiếp, không giao nhau → lấy hai đoạn có tổng nhỏ nhất, nhưng phải nằm ở 2 phía khác nhau.

- **Cách tìm đoạn có tổng nhỏ nhất:**

  + Duyệt tất cả các đoạn [L..R] và tính nhanh tổng bằng **mảng cộng dồn**.

- **Ghép 2 đoạn (trường hợp xóa 2 đoạn)**

  + Cắt mảng tại vị trí i:

    ▪ Bên trái (≤ i): lấy đoạn nhỏ nhất trong prefix.

    ▪ Bên phải (≥ i+1): lấy đoạn nhỏ nhất trong suffix.

    ▪ Cộng 2 đoạn này lại, tính trung bình phần còn lại, so sánh với kết quả tốt nhất.

- So sánh tất cả các khả năng (0 đoạn, 1 đoạn, 2 đoạn) → chọn giá trị trung bình lớn nhất.

2.3. Mô tả thuật toán:

- Bước 1. Đọc dữ liệu và khởi tạo

+ Đọc `n` và mảng `a`.

+ Tính `tong = sum(a)` => Giá trị tổng của toàn mảng `a`.

- Bước 2. Tạo mảng cộng dồn (prefix sum)

+ Tạo mảng `cong_don` độ dài `n+1` với mọi phần tử ban đầu đều có giá trị bằng `0`.

+ Với mỗi `i`: `cong_don[i+1] = cong_don[i] + a[i]`.

=> Dùng để tính nhanh tổng đoạn `[L..R]` bằng `tong_doan = cong_don[R+1] - cong_don[L]`.

- Bước 3. Khởi tạo các mảng phụ và kết quả tạm

+ `tot_nhat_ket_thuc[R]` lưu `(tong_bi_xoa, len_bi_xoa)` cho **đoạn tốt nhất kết thúc tại** `R` (ban đầu `(0,0)`).

+ `tot_nhat_bat_dau[L]` lưu `(tong_bi_xoa, len_bi_xoa)` cho **đoạn tốt nhất bắt đầu tại** `L` (ban đầu `(0,0)`).

+ `tu_so_tot, mau_so_tot = tong, n` => lưu tử số và mẫu số của phương án tốt nhất hiện tại (ban đầu là không xóa).

- Bước 4. Định nghĩa hàm so sánh hai phương án xóa

+ `phuong_an_xoa_tot_hon(tong1, len1, tong2, len2)` trả về `True` nếu: `(tong - tong1)/(n - len1) > (tong - tong2)/(n - len2)`.

+ Thực hiện bằng nhân chéo:

`(tong - tong1)*(n - len2) > (tong - tong2)*(n - len1)` để tránh chia số thực.

+ Hàm cũng kiểm tra `n - len > 0` (không cho xóa hết mảng).

- Bước 5. Duyệt mọi đoạn [L..R] (hai vòng for)

+ Với mỗi cặp `(L,R)`:

- Tính:

```
tong_doan = cong_don[R+1] - cong_don[L]
do_dai = R - L + 1.
```

- Bỏ qua nếu `n - do_dai <= 0` (không được xóa cả mảng).

+ Cập nhật phương án xóa 1 đoạn toàn cục:

- `tu_so = tong - tong_doan`
- `mau_so = n - do_dai`
- Nếu `tu_so/mau_so` tốt hơn `tu_so_tot/mau_so_tot` (so sánh chéo) → cập nhật:

```
tu_so_tot = tu_so
mau_so_tot = mau_so
```

+ Cập nhật mảng `tot_nhat_ket_thuc[R]`:

- Lấy `tong_bi_xoa, len_bi_xoa = tot_nhat_ket_thuc[R]`.
- Nếu xóa `[L..R]` tốt hơn xóa đoạn đang lưu → gán `tot_nhat_ket_thuc[R] = (tong_doan, do_dai)`.

+ Cập nhật mảng `tot_nhat_bat_dau[L]` tương tự (so sánh với phần đang lưu).

=> Kết quả sau bước này: với mỗi `R` ta biết đoạn kết thúc ở `R` nào là "ứng viên tốt nhất"; tương tự cho mỗi `L`.

- Bước 6. Xây mảng tốt nhất trong prefix

+ Tạo mảng `tot_nhat_trong_prefix[i]`: tốt nhất (theo trung bình phần còn lại) khi xóa **1** đoạn nằm hoàn toàn trong `[0..i]`.

+ Cách thực hiên: Quét **i** từ **trái** sang **phải**, giữ `doan_hien_tai_tot` là phần tốt nhất đến giờ; với mỗi `doan_dang_xet` so sánh `tot_nhat_ket_thuc[i]` với `doan_hien_tai_tot` và cập nhật rồi gán vào `tot_nhat_trong_prefix[i]`.

- Bước 7. Xây mảng tốt nhất trong suffix

+ Tạo mảng `tot_nhat_tu_suffix[i]`: tốt nhất khi xóa **1** đoạn nằm hoàn toàn trong `[i..n-1]`.

+ Cách thực hiện: Quét từ **phải** sang **trái**, tương tự bước prefix nhưng dùng `tot_nhat_bat_dau[i]`.

- Bước 8. Ghép 2 đoạn không giao nhau

+ Thử mọi vị trí cắt `i = 0..n-2` (chia thành **[0..i]** và **[i+1..n-1]**):

- Lấy `left_tong,left_len = tot_nhat_trong_prefix[i]`
- Lấy `right_tong,right_len = tot_nhat_tu_suffix[i+1]`
- Lấy tổng bị xóa: `tong_bi_xoa = left_tong + right_tong`
- Lấy độ dài bị xóa: `len_bi_xoa = left_len + right_len`
- Nếu `n - len_bi_xoa <= 0` thì bỏ qua.
- Tính `tu_so = tong - tong_bi_xoa, mau_so = n - len_bi_xoa`. Nếu tốt hơn `tu_so_tot/mau_so_tot` → cập nhật.

=> Như vậy ta xét mọi phép chọn 2 đoạn không chồng nhau bằng cách ghép đoạn tốt nhất trong prefix với đoạn tốt nhất trong suffix tại mỗi vị trí cắt.

- Bước 9. Xuất kết quả

+ Kết quả là `ketqua = tu_so_tot // mau_so_tot` (lấy phần nguyên)

+ Ghi ra file `TBCONG.OUT`.

2.4. Code tham khảo (Python và C++):

2.4.1. Code Python:

```
# ========================== ĐỌC DỮ LIỆU ==========================
fi = open("TBCONG.INP", "r")                    # Mở file để đọc
n = int(fi.readline().strip())                  # Nhập số lượng phần tử n
a = list(map(int, fi.readline().split()))       # Nhập mảng a
fi.close()                                      # Đóng file sau khi đọc
# ========================== BƯỚC 1: TÍNH TỔNG ==========================
```

```
# Tính tổng cả mảng a ban đầu
tong = sum(a)
# ========================== BƯỚC 2: MẢNG CỘNG DỒN ==========================
# Tạo mảng cộng dồn để tính nhanh tổng đoạn: cong_don[i] = Tổng a[0..i-1]
cong_don = [0] * (n + 1)
for i in range(0, n):
    cong_don[i+1] = cong_don[i] + a[i]
# ========================== BƯỚC 3: KHỞI TẠO MẢNG PHỤ ==========================
# Khởi tạo các mảng phụ ban đầu: tot_nhat_ket_thuc[R] và tot_nhat_bat_dau[L]
# tot_nhat_ket_thuc[R] = (tong_bi_xoa, len_bi_ xoa) của đoạn kết thúc tại R tốt nhất.
# Nghĩa là "xóa đoạn tốt nhất kết thúc tại R".
tot_nhat_ket_thuc = []
for i in range(0, n):
    tot_nhat_ket_thuc.append((0, 0))
# tot_nhat_bat_dau[L] = (tong_bi_xoa, len_bi_xoa) của đoạn bắt đầu tại L tốt nhất.
# Nghĩa là "xóa đoạn tốt nhất bắt đầu tại L"
tot_nhat_bat_dau = []
for i in range(0, n):
    tot_nhat_bat_dau.append((0, 0))
# Khởi tạo Kết quả ban đầu: Không xóa gì (tu_so_tot và mau_so_tot hiện tại)
tu_so_tot = tong
mau_so_tot = n
# ========================== BƯỚC 4: HÀM SO SÁNH ==========================
# Tạo hàm phuong_an_xoa_tot_hon để so sánh phương án xóa (tong1, len1) có cho trung
bình phần còn lại LỚN HƠN phương án (tong2, len2) hay không. Nghĩa là "hàm so sánh 2
phương án xóa".
def phuong_an_xoa_tot_hon(tong1, len1, tong2, len2):
    # Không được xóa hết mảng. Nếu xóa dẫn đến không còn phần tử -> không hợp lệ
    if n - len1 <= 0: return False
    if n - len2 <= 0: return True
    # So sánh (tong - tong1)/(n - len1) > (tong - tong2)/(n - len2) bằng nhân chéo
    return (tong - tong1) * (n - len2) > (tong - tong2) * (n -
len1)
# ========================== BƯỚC 5: DUYỆT MỌI ĐOẠN [L..R] ========================
for L in range(0, n):
    for R in range(L, n):
        tong_doan = cong_don[R+1] - cong_don[L] # Tổng đoạn [L..R]
        do_dai = R - L + 1                      # Độ dài đoạn [L..R] đang
xét
        # Bỏ qua trường hợp xóa toàn bộ mảng
        if n - do_dai <= 0: continue
        # Cập nhật phương án xóa 1 đoạn toàn cục
        tu_so = tong - tong_doan
        mau_so = n - do_dai
        if tu_so * mau_so_tot > tu_so_tot * mau_so:
            tu_so_tot = tu_so
            mau_so_tot = mau_so
        # Cập nhật mảng tot_nhat_ket_thuc[R] (chỉ xét đoạn kết thúc ở R)
        tong_bi_xoa, len_bi_xoa = tot_nhat_ket_thuc[R]
```

```
        if phuong_an_xoa_tot_hon(tong_doan, do_dai, tong_bi_xoa,
len_bi_xoa):
            tot_nhat_ket_thuc[R] = (tong_doan, do_dai)
        # Cập nhật mảng tot_nhat_bat_dau[L] (chỉ xét đoạn bắt đầu ở L)
        tong_bi_xoa, len_bi_xoa = tot_nhat_bat_dau[L]
        if phuong_an_xoa_tot_hon(tong_doan, do_dai, tong_bi_xoa,
len_bi_xoa):
            tot_nhat_bat_dau[L] = (tong_doan, do_dai)
# ========================== BƯỚC 6: TẠO PREFIX BEST ==========================
# tot_nhat_trong_prefix[i] = tốt nhất khi xóa 1 đoạn hoàn toàn trong [0..i].
# Nghĩa là "xóa đoạn tốt nhất nằm trong [0..i]"
tot_nhat_trong_prefix = [(0, 0)] * n
doan_hien_tai_tot = (0, 0)
for i in range(0, n):
    doan_dang_xet = tot_nhat_ket_thuc[i]
    if phuong_an_xoa_tot_hon(doan_dang_xet[0], doan_dang_xet[1],
doan_hien_tai_tot[0], doan_hien_tai_tot[1]):
        doan_hien_tai_tot = doan_dang_xet
    tot_nhat_trong_prefix[i] = doan_hien_tai_tot
# ========================== BƯỚC 7: TẠO SUFFIX BEST ==========================
# tot_nhat_tu_suffix[i] = tốt nhất khi xóa 1 đoạn hoàn toàn trong [i..n-1].
# Nghĩa là "xóa đoạn tốt nhất nằm trong [i..n-1]"
tot_nhat_tu_suffix = [(0, 0)] * n
doan_hien_tai_tot = (0, 0)
for i in range(n-1, -1, -1):
    doan_dang_xet = tot_nhat_bat_dau[i]
    if phuong_an_xoa_tot_hon(doan_dang_xet[0], doan_dang_xet[1],
doan_hien_tai_tot[0], doan_hien_tai_tot[1]):
        doan_hien_tai_tot = doan_dang_xet
    tot_nhat_tu_suffix[i] = doan_hien_tai_tot
# ========================== BƯỚC 8: GHÉP 2 ĐOẠN ==========================
# Ghép 2 đoạn: Thử cắt giữa i và i+1, lấy tốt nhất bên trái + tốt nhất bên phải
for i in range(0, n-1):
    left_tong, left_len = tot_nhat_trong_prefix[i]
    right_tong, right_len = tot_nhat_tu_suffix[i+1]
    tong_bi_xoa = left_tong + right_tong
    len_bi_xoa = left_len + right_len
    if n - len_bi_xoa <= 0: continue
    tu_so = tong - tong_bi_xoa
    mau_so = n - len_bi_xoa
    if tu_so * mau_so_tot > tu_so_tot * mau_so:
        tu_so_tot = tu_so
        mau_so_tot = mau_so
# ========================== BƯỚC 9: XUẤT KẾT QUẢ ==========================
ketqua = tu_so_tot // mau_so_tot
fo = open("TBCONG.OUT", "w")
fo.write(f'{ketqua}')
fo.close()
```

2.4.2. Code C++:

```
#include <iostream>
#include <bits/stdc++.h>
#include <fstream> // Khai bao cai nay de su dung ifstream va ofstream
#include <vector> // Khai bao cai nay de su dung vector
using namespace std;
int main()
{
    // ========================== DOC DU LIEU ==========================
    ifstream fi("TBCONG.INP");                  // Mo file de doc
    int n; fi >> n;                             // Nhap so luong phan tu n
    vector <long long> a(n);                    // Nhap mang a
    for (int i = 0; i < n; i++) fi >> a[i];
    fi.close();                                 // Dong file sau khi
doc
    // ========================== BUOC 1: TINH TONG ==========================
    // Tinh tong ca mang a ban dau
    long long tong = 0;
    for (int i = 0; i < n; i++) tong += a[i];
    // ========================== BUOC 2: MANG CONG DON ========================
    // Tao mang cong don de tinh nhanh tong đoạn: cong_don[i] = Tong a[0..i-1]
    vector <long long> cong_don(n+1, 0);
    for (int i = 0; i < n; i++)
    {
        cong_don[i+1] = cong_don[i] + a[i];
    }
    // ========================== BUOC 3: KHOI TAO MANG PHU ====================
    // Khoi tao cac mang phu ban dau: tot_nhat_ket_thuc[R] và tot_nhat_bat_dau[L]
    // tot_nhat_ket_thuc[R] = (tong_bi_xoa, len_bi_xoa) cua doan ket thuc tai R tot
nhat.
    // Nghia la "xoa doan tot nhat ket thuc tai R".
    vector <pair<long long, int>> tot_nhat_ket_thuc(n, {0,0});
    // tot_nhat_bat_dau[L] = (tong_bi_xoa, len_bi_xoa) cua doan bat dau tai L tot
nhat.
    // Nghia la "xoa doan tot nhat bat dau tai L".
    vector <pair<long long, int>> tot_nhat_bat_dau(n, {0,0});
    // Khoi tao Ket qua ban dau: Khong xoa gi (tu_so_tot va mau_so_tot hien tai)
    long long tu_so_tot = tong;
    int mau_so_tot = n;
    // ========================== BUOC 4: HAM SO SANH ========================
// Tao ham phuong_an_xoa_tot_hon de so sanh phuong an xoa (tong1, len1) co cho trung
binh phan con lai LON HON phuong an (tong2, len2) hay khong. Nghia la "ham so sanh 2
phuong an xoa".
    auto phuong_an_xoa_tot_hon = [&](long long tong1, int len1,
long long tong2, int len2)
    {
        // Khong duoc xoa het mang. Neu xoa dan den khong con phan tu -> khong hop le
        if (n - len1 <= 0) return false;
```

```
        if (n - len2 <= 0) return true;
        // So sanh (tong - tong1)/(n - len1) > (tong - tong2)/(n - len2) bang nhan
cheo
        return (tong - tong1) * (n - len2) > (tong - tong2) * (n -
len1);
    };
    // ========================== BUOC 5: DUYET MOI DOAN [L..R] ===============
    for (int L = 0; L < n; L++)
    {
        for (int R = L; R < n; R++)
        {
            long long tong_doan = cong_don[R+1] - cong_don[L];  //
Tong doan [L..R]
            int do_dai = R - L + 1;                             //
Do dai doan [L..R] dang xet
            // Bo qua truong hop xoa toan bo mang
            if (n - do_dai <= 0) continue;
            // Cap nhat phuong an xoa 1 doan toan cuc
            long long tu_so = tong - tong_doan;
            int mau_so = n - do_dai;
            if (tu_so * 1LL * mau_so_tot > tu_so_tot * 1LL *
mau_so)
            {
                tu_so_tot = tu_so;
                mau_so_tot = mau_so;
            }
            // Cap nhat mang tot_nhat_ket_thuc[R] (chi xet doan ket thuc o R)
            auto [tong_bi_xoa, len_bi_xoa] = tot_nhat_ket_thuc[R];
            if (phuong_an_xoa_tot_hon(tong_doan, do_dai,
tong_bi_xoa, len_bi_xoa))
            {
                tot_nhat_ket_thuc[R] = {tong_doan, do_dai};
            }
            // Cap nhat mang tot_nhat_bat_dau[L] (chi xet doan bat dau o L)
            auto [tong_bi_xoa2, len_bi_xoa2] = tot_nhat_bat_dau[L];
            if (phuong_an_xoa_tot_hon(tong_doan, do_dai,
tong_bi_xoa2, len_bi_xoa2))
            {
                tot_nhat_bat_dau[L] = {tong_doan, do_dai};
            }
        }
    }
    // ========================== BUOC 6: TAO PREFIX BEST ====================
    // tot_nhat_trong_prefix[i] = tot nhat khi xoa 1 doan hoan toan trong [0..i].
    // Nghia la "xoa doan tot nhat nam trong [0..i]".
    vector <pair<long long,int>> tot_nhat_trong_prefix(n, {0,0});
    pair <long long,int> doan_hien_tai_tot = {0,0};
```

```
    for (int i = 0; i < n; i++)
    {
        auto doan_dang_xet = tot_nhat_ket_thuc[i];
        if (phuong_an_xoa_tot_hon(doan_dang_xet.first,
doan_dang_xet.second, doan_hien_tai_tot.first,
doan_hien_tai_tot.second))
        {
            doan_hien_tai_tot = doan_dang_xet;
        }
        tot_nhat_trong_prefix[i] = doan_hien_tai_tot;
    }
    // ============================ BUOC 7: TAO SUFFIX BEST =====================
    // tot_nhat_tu_suffix[i] = tot nhat khi xoa 1 doan hoan toan trong [i..n-1].
    // Nghia la "xoa doan tot nhat nam trong [i..n-1]".
    vector <pair<long long,int>> tot_nhat_tu_suffix(n, {0,0});
    doan_hien_tai_tot = {0,0};
    for (int i = n-1; i >= 0; i--)
    {
        auto doan_dang_xet = tot_nhat_bat_dau[i];
        if (phuong_an_xoa_tot_hon(doan_dang_xet.first,
doan_dang_xet.second, doan_hien_tai_tot.first,
doan_hien_tai_tot.second))
        {
            doan_hien_tai_tot = doan_dang_xet;
        }
        tot_nhat_tu_suffix[i] = doan_hien_tai_tot;
    }
    // ============================ BUOC 8: GHEP 2 DOAN ========================
    // Ghep 2 doan: Thu cat giua i và i+1, lay tot nhat ben trai + tot nhat ben phai
    for (int i = 0; i < n-1; i++)
    {
        long long left_tong = tot_nhat_trong_prefix[i].first;
        int left_len = tot_nhat_trong_prefix[i].second;
        long long right_tong = tot_nhat_tu_suffix[i+1].first;
        int right_len = tot_nhat_tu_suffix[i+1].second;
        long long tong_bi_xoa = left_tong + right_tong;
        int len_bi_xoa = left_len + right_len;
        if (n - len_bi_xoa <= 0) continue;
        long long tu_so = tong - tong_bi_xoa;
        int mau_so = n - len_bi_xoa;
        if (tu_so * 1LL * mau_so_tot > tu_so_tot * 1LL * mau_so)
        {
            tu_so_tot = tu_so;
            mau_so_tot = mau_so;
        }
    }
```

```
    // ========================== BUOC 9: XUAT KET QUA ========================
    long long ketqua = tu_so_tot / mau_so_tot;
    ofstream fo("TBCONG.OUT");
    fo << ketqua;
    fo.close();
    return 0;
}
```

3. BÀI 3. BỘ BA SỐ *(2 điểm)*

BÀI 3. BỘ BA SỐ *(2 điểm)*

Cho n số nguyên không âm $a_1, a_2, \dots, a_n$ và một số nguyên dương m. Hãy đếm số bộ ba số (i, j, k) mà $a_i * a_j * a_k$ chia hết cho m (lưu ý nếu 2 bộ ba mà bộ này là hoán vị của bộ kia thì vẫn tính là 2 bộ, ví dụ $(1, 2, 3)$ và $(2, 1, 3)$ là hai bộ khác nhau).

Dữ liệu vào: Đọc từ file văn bản BOBASO.INP gồm:

- Dòng đầu tiên chứa 2 số nguyên n và $m (1 \le n \le 2.10^3, 1 \le m \le 3.10^3)$.
- Dòng thứ hai chứa n số nguyên không âm $a_1, a_2, \dots, a_n (0 \le a_i \le 10^9)$.

Kết quả: Ghi ra file văn bản BOBASO.OUT một dòng là số bộ ba số thỏa mãn yêu cầu.

Ví dụ:

BOBASO.INP	BOBASO.OUT	Giải thích
2 5 1 5	7	Có 7 bộ ba là (1,1,5), (1,5,1), (1,5,5), (5,1,1), (5,1,5), (5,5,1), (5,5,5)
10 3 1 2 3 4 5 6 7 8 9 10	657	Có tất cả 657 bộ ba số thỏa mãn chia hết cho 3.

Ràng buộc:

- Có 50% số test với $1 \le n \le 200$;
- Có 50% số test còn lại với $200 < m \le 2.10^3$.

3.1. Tóm tắt đề bài:

- Cho dãy số $a_1, a_1, a_3, \dots, a_n$ và số nguyên dương m. Hãy đếm số bộ ba có thứ tự (i, j, k) sao cho: $a_i * a_j * a_k$ chia hết cho m.
- Lưu ý: Bộ ba có thứ tự nghĩa là (1, 2, 3) khác với (2, 1, 3)

3.2. Ý tưởng giải:

- Mệnh đề: Nếu ai * aj * ak chia hết cho m, thì UCLN(ai, m) * UCLN(aj, m) * UCLN(ak, m) cũng chia hết cho m và ngược lại.
- Bước 1. Nhập n, m và mảng a gồm n phần tử.
- Bước 2. Tính ƯCLN của từng phần tử a[i] với m, gom nhóm theo ƯCLN.
- Bước 3. Mỗi nhóm chỉ cần biết số lượng phần tử.
- Bước 4. Duyệt mọi bộ ba nhóm khác nhau.
- Bước 5. Nếu tích các ƯCLN chia hết cho m → cộng số cách chọn.
- Bước 6. Xuất kết quả cuối cùng.

3.3. Mô tả thuật toán:

- Bước 1. Chuẩn bị

+ Viết hàm UCLN(a,b)để tính ƯCLN của hai số.

- Bước 2. Nhập dữ liệu

+ Đọc vào n, m và mảng a[1..n].

- Bước 3. Gom nhóm theo ƯCLN

+ Khởi tạo từ điển rỗng `DEM (DEM = {})`.

+ Với mỗi phần tử a[i] trong mảng:

  - Tính `UOC = UCLN(a[i], m)`.

- Nếu `UOC` đã có trong `DEM` thì tăng giá trị đếm lên 1.
- Nếu chưa có thì thêm mới vào với giá trị 1.

+ Kết quả: `DEM` lưu dạng (**UOC : Số lượng phần tử có UCLN = UOC**).

- Bước 4. Lấy danh sách các nhóm ƯCLN

+ Tạo danh sách `NHOM_UCLN = list(DEM.keys())`.

+ Đây chính là các ƯCLN khác nhau xuất hiện trong mảng.

- Bước 5. Đếm bộ ba

+ Khởi tạo `ketqua = 0`.

+ Duyệt ba vòng for lồng nhau trên `NHOM_UCLN`:

- Với mỗi bộ ba `(UOC1, UOC2, UOC3)`:
  - Nếu (UOC1 * UOC2 * UOC3) mod m = 0 thì cộng vào kết quả: ketqua = ketqua + DEM[UOC1] * DEM[UOC2] * DEM[UOC3]

- Bước 6. Xuất kết quả

+ Ghi `ketqua` ra file `BOBASO.OUT`.

3.4. Code tham khảo (Python và C++):

3.4.1. Code Python:

```python
# Bước 1. Viết hàm tìm ƯCLN(a, b)
def UCLN(a, b):
    if a == 0: return b
    return UCLN(b % a, a)
# Bước 2. Nhập dữ liệu ban đầu
fi = open('BOBASO.INP', 'r')            # Mở file để nhập
n, m = fi.readline().split()            # Nhập n, m
n, m = int(n), int(m)
a = list(map(int, fi.read().split()))   # Nhập mảng a
fi.close()                              # Đóng file sau khi nhập xong
# Bước 3. Gom nhóm theo ƯCLN
# Tìm UCLN của từng phần tử a[i] với m, UOC = UCLN(a[i], m).
# Sau đó lưu lại UOC và SO LUONG (UOC : SO LUONG) trong danh sách dict DEM{}
DEM = {} # Trong mỗi phần tử của dict DEM sẽ có 2 giá trị là (UOC : SO LUONG)
for i in range(0, n):
    UOC = UCLN(a[i], m)
    if UOC in DEM: DEM[UOC] = DEM[UOC] + 1
    else: DEM[UOC] = 1
# Bước 4. Lấy danh sách các nhóm ƯCLN. Đây chính là các ƯCLN khác nhau xuất hiện trong
mảng.
NHOM_UCLN = list(DEM.keys())
# Bước 5. Đếm bộ ba
ketqua = 0
# Duyệt 3 vòng lặp for lồng nhau trong danh sách NHOM_UCLN
# Nếu tích của các UOC chia hết cho m, thì tích SO LUONG của các UOC này cũng chia hết
cho m
for UOC1 in NHOM_UCLN:
    for UOC2 in NHOM_UCLN:
        for UOC3 in NHOM_UCLN:
            if (UOC1 * UOC2 * UOC3 % m == 0):
```

```
                ketqua = ketqua + DEM[UOC1] * DEM[UOC2] * DEM[UOC3]
# Bước 6. Xuất kết quả
fo = open('BOBASO.OUT', 'w')
fo.write(f'{ketqua}')
fo.close()
```

3.4.2. Code C++:

```
#include <iostream>
#include <bits/stdc++.h>
#include <fstream>  // Khai bao cai nay de su dung ifstream va ofstream
#include <vector>   // Khai bao cai nay de su dung vector
using namespace std;
// Buoc 1. Viet ham tim UCLN(a, b)
int UCLN(int a, int b)
{
    if (a == 0) return b;
    return UCLN(b % a, a);
}
int main()
{
    // Buoc 2. Nhap du lieu ban dau
    ifstream fi("BOBASO.INP");                // Mo file de nhap
    int n, m;
    fi >> n >> m;                             // Nhap n, m
    vector <int> a(n);                        // Khai bao mang a
    for (int i = 0; i < n; i++) fi >> a[i]; // Nhap mang a
    fi.close();                               // Dong file sau khi nhap xong
    // Buoc 3. Gom nhom theo UCLN
    // Tim UCLN cua tung phan tu a[i] voi m, UOC = UCLN(a[i], m).
    // Sau do luu lai UOC va SO LUONG (UOC : SO LUONG) trong danh sach map <int, int>
DEM
    map <int, int> DEM; // Trong moi phan tu cua map <int, int> DEM se co 2 gia
tri la (UOC : SO LUONG)
    for (int i = 0; i < n; i++)
    {
        int UOC = UCLN(a[i], m);
        if (DEM.find(UOC) != DEM.end()) DEM[UOC] = DEM[UOC] + 1;
        else DEM[UOC] = 1;
    }
    // Buoc 4. Lay danh sach cac nhom UCLN. Day chinh la cac UCLN khac nhau xuat hien
trong mang.
    vector <int> NHOM_UCLN;
    for (auto it : DEM)
    {
        NHOM_UCLN.push_back(it.first);
    }
    // Buoc 5. Dem bo ba
    long long ketqua = 0;
```

```
    // Duyet 3 vong lap for long nhau trong danh sach NHOM_UCLN
    // Neu tich cua cac UOC chia het cho m, thi tich SO LUONG cua cac UOC nay cung
chia het cho m
    for (int i = 0; i < (int)NHOM_UCLN.size(); i++)
    {
        for (int j = 0; j < (int)NHOM_UCLN.size(); j++)
        {
            for (int k = 0; k < (int)NHOM_UCLN.size(); k++)
            {
                int UOC1 = NHOM_UCLN[i];
                int UOC2 = NHOM_UCLN[j];
                int UOC3 = NHOM_UCLN[k];
                if ((1LL * UOC1 * UOC2 * UOC3) % m == 0)
                {
                    ketqua = ketqua + 1LL * DEM[UOC1] * DEM[UOC2] *
DEM[UOC3];
                }
            }
        }
    }
    // Buoc 6. Xuat ket qua
    ofstream fo("BOBASO.OUT");
    fo << ketqua;
    fo.close();
    return 0;
}
```

4. BÀI 4. RÚT TIỀN *(2 điểm)*

BÀI 4: RÚT TIỀN ***(2 điểm)***

An có rất nhiều tiền trong ngân hàng Thụy Sĩ, một hôm An cần rút một số tiền $N (N \leq 10^5)$, ngân hàng chỉ có $K (K \leq 10^3)$ loại mệnh giá lần lượt là $A_1, A_2, \ldots, A_K$. Vì lí do đặc biệt nên An mong muốn số tờ tiền rút được là ít nhất.

Dữ liệu vào: Đọc từ file văn bản TIEN.INP gồm:

- Dòng thứ nhất chứa 2 số nguyên dương N và K. Trong đó N là số tiền cần rút, K là loại tiền mệnh giá.
- Dòng thứ hai chứa K số nguyên dương $A_1, A_2, \ldots, A_K$ lần lượt là mệnh giá của các tờ tiền.

Kết quả: Ghi ra file văn bản TIEN.OUT chứa 1 số nguyên dương duy nhất là số tờ tiền ít nhất mà An rút được, nếu không thể in ra -1

Ví dụ:

TIEN.INP	TIEN.OUT	Giải thích
125 6 1 2 5 10 20 50	4	Số tờ tiền ít nhất có thể lấy là 4 tờ gồm 2 tờ mệnh giá 50, 1 tờ mệnh giá 20, một tờ mệnh giá 5.
5 3 2 4 6	-1	Không có cách nào để từ các tờ tiền mệnh giá 2 4 6 tạo thành số tiền là 5 cho nên ta in ra -1.

Ràng buộc:

- Có 20% số test có $K = 2$ và A_i khác nhau từng đôi một;
- Có 30% số test tiếp theo có $K \leq 10, N \leq 100$ và A_i khác nhau từng đôi một;
- Có 50% số test còn lại không có giới hạn gì khác.

4.1. Tóm tắt đề bài:

- Cho một số tiền cần rút N.
- Có K loại mệnh giá tiền A1,A2,…,AK.
- Yêu cầu: Rút ra đúng N đồng với **số tờ ít nhất**.
- Nếu không thể rút đúng số tiền N thì in ra **-1**.

4.2. Ý tưởng giải:

- Bài toán thuộc dạng **quy hoạch động (dp)**.
- Ta tính dần từ số tiền nhỏ đến số tiền lớn:

  `dp[x]` = số tờ ít nhất để rút ra đúng `x` đồng.
- Ban đầu: `dp[0] = 0` (0 đồng cần 0 tờ).
- Với mỗi số tiền `x`, thử cộng thêm từng mệnh giá. Nếu tìm được cách tốt hơn thì cập nhật bằng cách sử dụng công thức quy hoạch động sau:

  dp[x] = min(dp[x], dp[x – menh_gia] + 1), khi x ≥ menh_gia
- Kết quả cuối cùng là `dp[N]`. Nếu vẫn bằng "không xác định" thì in ra `-1`.

4.3. Mô tả thuật toán:

- Bước 1. Nhập dữ liệu

+ Đọc `N, K`.

+ Đọc danh sách `cac_menh_gia`. (Đã thay mảng A của đề bài thành mảng `cac_menh_gia`)

- Bước 2. Khởi tạo

+ Tạo một mảng `dp` có kích thước `N+1`.

+ Gán tất cả giá trị bằng một số rất lớn `khong_the_rut_duoc` (biểu diễn "vô cùng").

+ Gán `dp[0] = 0` vì 0 đồng thì cần 0 tờ.

- Bước 3. Xây dựng bảng `dp`

+ Với mỗi số tiền `so_tien_rut` chạy từ 1 đến N:
  - Với mỗi `menh_gia` trong `cac_menh_gia`:
    - Nếu `so_tien_rut >= menh_gia` thì cập nhật theo công thức quy hoạch động sau:

      `dp[so_tien_rut] = min(dp[so_tien_rut], dp[so_tien_rut - menh_gia] + 1)`

- Bước 4. Lấy kết quả

+ Nếu `dp[N] == khong_the_rut_duoc` thì in ra `-1`.

+ Ngược lại, in ra `dp[N]`.

- Bước 5. Xuất kết quả ra file TIEN.OUT

"Số tờ hiện tại" = giá trị dp[so_tien_rut] (tốt nhất ta biết cho tới lúc này).

"Số tờ thêm vào" = dp[so_tien_rut - menh_gia] + 1 (khi thử thêm 1 tờ mới).

"Cái nào tốt hơn thì lấy" = dùng min(...).

4.4. Code tham khảo (Python và C++):

4.4.1. Code Python:

```
# Bước 1. Nhập dữ liệu
fi = open('TIEN.INP', 'r')                          # Mở file để đọc
N, K = fi.readline().split()                        # Nhập N, K. N là số
tiền cần rút, K là số lượng mệnh giá
N, K = int(N), int(K)
cac_menh_gia = list(map(int, fi.read().split()))    # Nhập mảng a: Mảng
a được thay thế bằng mảng cac_menh_gia
fi.close()                                          # Đóng file sau khi
nhập
```

```
# Bước 2. Khởi tạo
khong_the_rut_duoc = 10**9            # Khai báo giá trị khong_the_rut_duoc
(giá trị vô cực)
dp = [khong_the_rut_duoc] * (N + 1) # Tạo mảng dp với kích thước N+1, khởi
tạo tất cả bằng khong_the_rut_duoc
dp[0] = 0                             # 0 đồng thì cần 0 tờ
# Bước 3. Xây dựng mảng dp
# Duyệt 2 vòng lặp for lồng nhau để tìm kết quả
# Vòng for 1: Duyệt so_tien_rut từ 1 đến N + 1
# Vòng for 2: Duyệt tất cả các menh_gia có trong bảng cac_menh_gia
# So sánh so_tien_rut và menh_gia. Nếu (so_tien_rut >= menh_gia) thì sử dụng công thức
quy hoạch động sau để tính kết quả:
# dp[so_tien_rut] = min(dp[so_tien_rut], dp[so_tien_rut - menh_gia] + 1)
for so_tien_rut in range(1, N + 1):
    for menh_gia in cac_menh_gia:
        if so_tien_rut >= menh_gia:
            dp[so_tien_rut] = min(dp[so_tien_rut], dp[so_tien_rut -
menh_gia] + 1)
# Bước 4. Lấy kết quả
if dp[N] == khong_the_rut_duoc: ketqua = -1
else: ketqua = dp[N]
# Bước 5. Xuất kết quả ra file TIEN.OUT
fo = open('TIEN.OUT', 'w')
fo.write(f'{ketqua}')
fo.close()
```

4.4.2. Code C++:

```
#include <iostream>
#include <bits/stdc++.h>
#include <fstream>  // Khai bao cai nay de su dung ifstream va ofstream
#include <vector>   // Khai bao cai nay de su dung vector
using namespace std;
int main()
{
    // Buoc 1. Nhap du lieu
    ifstream fi("TIEN.INP");          // Mo file de doc du lieu vao
    int N, K;
    fi >> N >> K;                     // Nhap N, K. N la so tien can rut, K la
so luong menh gia
    vector <int> cac_menh_gia(K);     // Nhap mang a: Mang a duoc thay the
bang mang cac_menh_gia
    for (int i = 0; i < K; i++)
    {
        fi >> cac_menh_gia[i];
    }
    fi.close();                       // Dong file sau khi nhap
    // Buoc 2. Khoi tao
    int khong_the_rut_duoc = 1000000000;          // Khai bao gia tri
khong_the_rut_duoc (gia tri vo cuc)
```

```
    vector <int> dp(N + 1, khong_the_rut_duoc); // Tao mang dp voi kich
thuoc N+1, khoi tao tat ca bang khong_the_rut_duoc
    dp[0] = 0;                                  // 0 dong thi can 0 to
    // Buoc 3. Xay dung mang dp
    // Duyet 2 vong lap for long nhau de tim ket qua
    // Vong for 1: Duyet so_tien_rut tu 1 đen N + 1
    // Vong for 2: Duyet tat ca cac menh_gia co trong bang cac_menh_gia
    // So sanh so_tien_rut va menh_gia. Neu (so_tien_rut >= menh_gia) thi su dung cong
thuc quy hoach dong sau de tinh ket qua:
    // dp[so_tien_rut] = min(dp[so_tien_rut], dp[so_tien_rut - menh_gia] + 1)
    for (int so_tien_rut = 1; so_tien_rut <= N; so_tien_rut++)
    {
        for (int menh_gia : cac_menh_gia)
        {
            if (so_tien_rut >= menh_gia)
            {
                dp[so_tien_rut] = min(dp[so_tien_rut],
dp[so_tien_rut - menh_gia] + 1);
            }
        }
    }
    // Buoc 4. Lay ket qua
    int ketqua;
    if (dp[N] == khong_the_rut_duoc) ketqua = -1;
    else ketqua = dp[N];
    // Buoc 5. Xuat ket qua ra file TIEN.OUT
    ofstream fo("TIEN.OUT");
    fo << ketqua;
    fo.close();
    return 0;
}
```

5. Các bộ Test:

Mỗi câu của đề thi đều có đủ 10 bộ Test, với các giá trị đầu vào tương ứng với tỉ lệ (%) của các ràng buộc (Substack) mà đề bài yêu cầu. Những bộ Test này đã được gửi ở tệp tin đính kèm cùng với tài liệu này.

6. Kết quả chấm thử bằng phần mềm Themis 1.9, giới hạn thời gian chạy của mỗi bộ Test là 01 giây:

Tên bài	Điểm tối đa	Điểm chấm thử bằng Themis 1.9	
		Python	C++
BÀI 1. CHUẨN HÓA (CHUANHOA.*)	3.0 điểm	3.0 điểm (100%)	3.0 điểm (100%)
BÀI 2. TRUNG BÌNH CỘNG (TBCONG.*)	3.0 điểm	1.5 điểm (50%)	2.4 điểm (80%)
BÀI 3. BỘ BA SỐ (BOBASO.*)	2.0 điểm	2.0 điểm (100%)	2.0 điểm (100%)
BÀI 4. RÚT TIỀN (TIEN.*)	2.0 điểm	1.2 điểm (60%)	2.0 điểm (100%)
Tổng số điểm đạt được:		**7.7 điểm (77%)**	**9.4 điểm (94%)**

ĐỀ SỐ 5
(THPT Chuyên Lê Quý Đôn – Đà Nẵng – Năm học: 2022 – 2023)

SỞ GIÁO DỤC VÀ ĐÀO TẠO
THÀNH PHỐ ĐÀ NẴNG

KỲ THI TUYỂN SINH LỚP 10
TRUNG HỌC PHỔ THÔNG NĂM HỌC 2022-2023

ĐỀ CHÍNH THỨC
(Đề thi có 02 trang)

Môn thi: **TIN HỌC** (chuyên)
Thời gian: **150** phút (không kể thời gian giao đề)
(Dành cho thí sinh thi vào Trường THPT chuyên Lê Quý Đôn)

TỔNG QUAN ĐỀ THI

Bài	Tên bài	File chương trình	Dữ liệu vào	Dữ liệu ra	Điểm
1	KHOẢNG CÁCH	KC.*	Bàn phím	Màn hình	4
2	BÁN DƯA	DUA.*	DUA.INP	DUA.OUT	3
3	SỐ CHỮ SỐ	SCS.*	SCS.INP	SCS.OUT	3

*(Phần mở rộng * là PAS hoặc CPP hoặc PY tùy theo ngôn ngữ và môi trường lập trình)*

Bài 1. KHOẢNG CÁCH

Trên một con đường có bố trí sẵn một số bóng đèn. Sơ đồ bố trí các bóng đèn trên con đường được biểu diễn bằng một xâu kí tự gồm kí tự chữ cái D đại diện cho bóng đèn và giữa hai kí tự chữ cái D có một vài kí tự chữ số biểu diễn cho một số nguyên dương là khoảng cách giữa hai bóng đèn này (đơn vị mét).

Yêu cầu: Xác định khoảng cách (đơn vị mét) giữa bóng đèn đầu tiên và bóng đèn cuối cùng trên con đường.

Input: Nhập từ bàn phím một xâu kí tự biểu diễn sơ đồ bố trí bóng đèn.

Output: In ra màn hình một số nguyên m theo yêu cầu.

Ví dụ:

Nhập từ bàn phím	In ra màn hình
D3D2D10D	15

Giới hạn: Xâu kí tự nhập từ bàn phím có không quá 255 kí tự và số nguyên giữa 2 kí tự D không vượt quá 1000.

Bài 2. BÁN DƯA

Năm nay được mùa dưa, Tý theo bố mang rất nhiều quả dưa ra chợ bán. Có một ông thương gia đến chợ mua dưa, biết Tý là người thông minh, ông ta muốn đưa ra thử thách cho Tý. Ông yêu cầu Tý thực hiện theo đúng trình tự sau:

Bước 1: Tý phải chọn ra N viên đá từ đống đá có sẵn ở chợ.

Bước 2: Trong N viên đá được chọn, loại ra những viên đá có khối lượng giống nhau.

Bước 3: Ông ta sẽ mua những quả dưa có tổng khối lượng đúng bằng tổng khối lượng của những viên đá còn lại sau khi thực hiện bước 2.

Tý đã rất nhanh chóng thực hiện yêu cầu trên và bán được số dưa có tổng khối lượng lớn nhất.

Yêu cầu: Em hãy tìm ra cách Tý đã làm.

Input: Đọc ở file văn bản **DUA.INP** có cấu trúc như sau:

- Dòng thứ nhất chứa 2 số nguyên dương N và K ($N \le K$), mỗi số cách nhau một dấu cách.
- Dòng thứ hai chứa K số nguyên dương, mỗi số cách nhau một dấu cách. Mỗi số trong dãy số này biểu thị cho khối lượng của mỗi viên đá và mỗi số không vượt quá 10000.
- Số lượng các viên đá có khối lượng nhỏ không ít hơn số lượng các viên đá có khối lượng lớn hơn.

Output: Ghi ra file văn bản **DUA.OUT** một số nguyên m là khối lượng dưa lớn nhất mà bố của Tý bán được.

Ví dụ:

DUA.INP	DUA.OUT
8 14 1 7 9 6 4 3 12 6 7 3 4 1 1 3	41

Giải thích: Tý chọn 8 viên đá có khối lượng là 1; 7; 9; 6; 4; 3; 12; 1 sau đó bỏ đi 2 viên có khối lượng bằng 1 thì tổng khối lượng còn lại là 41.

Giới hạn: 80% test ứng $K \le 10^4$; 20% test ứng với $10^4 < K \le 10^5$.

Bài 3. SỐ CHỮ SỐ

Người ta viết liên tiếp các số chính phương 1^2; 2^2; 3^2; 4^2; 5^2; …; k^2 thì được một số tự nhiên x = 149162536…

Yêu cầu: Xác định số lượng các chữ số của x.

Input: Đọc từ file văn bản **SCS.INP** một số nguyên dương k.

Output: Ghi ra file văn bản **SCS.OUT** số nguyên m theo yêu cầu.

Ví dụ:

SCS.INP	SCS.OUT
9	15

Giải thích: Với k = 9 ta có x = 149162536496481 là số tự nhiên có 15 chữ số.

Giới hạn: 30% test ứng với $k \le 10^6$; 40% test ứng với $10^6 < k \le 10^9$; 30% test ứng với $10^9 < k \le 10^{12}$.

----- Hết -----

Họ và tên thí sinh	*Phòng thi số*	*Số báo danh*

BÀI GIẢI ĐỀ SỐ 5
(THPT Chuyên Lê Quý Đôn – Đà Nẵng – Năm học: 2022 – 2023)

1. Bài 1. KHOẢNG CÁCH

Trên một con đường có bố trí sẵn một số bóng đèn. Sơ đồ bố trí các bóng đèn trên con đường được biểu diễn bằng một xâu kí tự gồm kí tự chữ cái D đại diện cho bóng đèn và giữa hai kí tự chữ cái D có một vài kí tự chữ số biểu diễn cho một số nguyên dương là khoảng cách giữa hai bóng đèn này (đơn vị mét).

Yêu cầu: Xác định khoảng cách (đơn vị mét) giữa bóng đèn đầu tiên và bóng đèn cuối cùng trên con đường.

Input: Nhập từ bàn phím một xâu kí tự biểu diễn sơ đồ bố trí bóng đèn.

Output: In ra màn hình một số nguyên m theo yêu cầu.

Ví dụ:

Nhập từ bàn phím	In ra màn hình
D3D2D10D	15

Giới hạn: Xâu kí tự nhập từ bàn phím có không quá 255 kí tự và số nguyên giữa 2 kí tự D không vượt quá 1000.

1.1. Tóm tắt đề bài:

- Nhập vào một xâu ký tự **s** bất kỳ.
- Tách tất cả các số nguyên liên tiếp xuất hiện trong xâu **s** vừa nhập.
- Tính tổng **m** của các số nguyên vừa tách được.
- In ra tổng **m**.

1.2. Ý tưởng giải:

- Nhập vào một xâu ký tự **s**.
- Sử dụng vòng lặp **for** để duyệt từng ký tự **s[i]** trong xâu **s**:
  + Nếu **s[i]** là ký tự số, ghép vào xâu tạm **s1** để tạo ra số nguyên liên tiếp.
  + Ngược lại, nếu ký tự **s[i]** không phải là số, gán xâu tạm **s1** vừa tạo ra trước đó vào trong mảng **a**, sau đó đặt lại xâu tạm **s1** thành rỗng.
- Sử dụng vòng lặp **for** để tính tổng **m** các phần tử **a[i]** là số ở trong mảng **a**.
- In ra tổng **m**.

1.3. Mô tả thuật toán:

- **Bước 1.** Nhập vào một xâu ký tự **s**.
- **Bước 2**. Thêm một ký tự không phải số vào cuối xâu **s** để đảm bảo số cuối cùng trong xâu s được tách ra (Ví dụ: **s = s + 'D'**)
- **Bước 3.** Khởi tạo mảng **a** chứa các số liên tiếp được tách ra từ xâu **s** ban đầu.
- **Bước 4.** Khởi tạo xâu tạm rỗng **s1 = ''** để lưu lại các số được tách ra từ xâu **s.**
- **Bước 5.** Sử dụng vòng lặp **for** để duyệt và tách từng ký tự số trong xâu **s**:
  + Nếu **s[i]** đang xét là ký tự số, thì ghép **s[i]** vào xâu tạm **s1** để tạo ra số nguyên liên tiếp.
  + Ngược lại, nếu ký tự **s[i]** không phải là số, thì gán xâu tạm **s1** vừa tạo ra trước đó vào trong mảng **a**, sau đó đặt lại xâu tạm **s1** thành rỗng để lưu số tiếp theo.

- **Bước 6.** Khởi tạo biến **m** để lưu lại kết quả theo yêu cầu của đề bài.
- **Bước 7.** Sử dụng vòng lặp **for** duyệt từng phần tử **a[i]** trong mảng **a** để tính tổng **m**:

Nếu phần tử **a[i]** của mảng **a** là số thì: Chuyển phần tử **a[i]** thành **số nguyên**, rồi cộng dồn **a[i]** vào biến tổng **m.**

- **Bước 6.** In ra tổng **m** các số.

1.4. Code tham khảo (Python và C++):

1.4.1. Code Python:

```
# Nhap vao xau ky tu s ban dau
# s = str(input())
fi = open('KC.INP', 'r')
s = fi.read()
s = str(s)
fi.close()
# Them 1 ky tu khong phai so vao cuoi xau s de dam bao so cuoi cung trong xau s duoc
tach ra
s = s + 'D'
# Khoi tao mang a chua cac ki tu so lien tiep duoc tach ra tu xau s
a = []
# Khoi tao xau s1 bang rong de luu cac so duoc tach ra tu xau s ban dau
s1 = ''
# Su dung vong lap for de duyet va tach ra tung so trong xau s ban dau
for i in range(0, len(s)):
    # Neu ky tu dang xet la so thi them vao xau s1
    if '0' <= s[i] <= '9': s1 = s1 + s[i]
    else:
        # Neu s[i] khong phai so thi them s1 vao mang a
        a.append(s1)
        # Khoi tao lai s1 bang rong de luu so tiep theo
        s1 = ''
# Su dung vong lap for de tinh tong m cac so trong mang a
# Khoi tao bien m de luu tong cac so trong mang a
m = 0
for i in range(0, len(a)):
    # Neu phan tu a[i] khac xau rong
    if a[i] != '':
        # Chuyen doi phan tu a[i] tu kieu string sang kieu int
        a[i] = int(a[i])
        # Tinh tong m
        m = m + a[i]
# In ra tong m cac so
# print(m)
fo = open('KC.OUT', 'w')
fo.write(f'{m}')
fo.close()
```

1.4.2. Code C++:

```
#include <iostream>
#include <bits/stdc++.h>
#include <fstream> // Khai bao cai nay de su dung ifstream va ofstream
#include <string> // Khai bao cai nay de su dung duoc string
#include <vector> // Khai bao cai nay de su dung duoc vector
using namespace std;
int main()
{
    // Nhap vao xau ky tu s ban dau
    // string s;
    // getline(cin, s);
    ifstream fi("KC.INP");
    string s;
    fi >> s;
    fi.close();
    // Them 1 ky tu khong phai so vao cuoi xau s de dam bao so cuoi cung trong xau s
duoc tach ra
    s = s + 'D';
    // Khoi tao mang a chua cac ki tu so lien tiep duoc tach ra tu xau s
    vector <string> a;
    // Khoi tao xau s1 bang rong de luu cac so duoc tach ra tu xau s ban dau
    string s1 = "";
    // Su dung vong lap for de duyet va tach ra tung so trong xau s ban dau
    for (int i = 0; i < s.length(); i++)
    {
        // Neu ky tu dang xet la so thi them vao xau s1
        if (s[i] >= '0' && s[i] <= '9') s1 = s1 + s[i];
        else
        {
            // Neu s[i] khong phai so thi them s1 vao mang a
            a.push_back(s1);
            // Khoi tao lai s1 bang rong de luu so tiep theo
            s1 = "";
        }
    }
    // Su dung vong lap for de tinh tong m cac so trong mang a
    // Khoi tao bien m de luu tong cac so trong mang a
    int m = 0;
    // Khoi tao bien tam de luu gia tri a[i] chuyen doi tu string sang int
    int tam;
    for (int i = 0; i < a.size(); i++)
    {
        // Neu phan tu a[i] khac xau rong
        if (a[i] != "")
        {
```

```
            // Chuyen doi phan tu a[i] tu kieu string sang kieu int va luu vao bien tam
            tam = stoi(a[i]);
            // Tinh tong m
            m = m + tam;
        }
    }
    // In ra tong m cac so
    // cout << m << endl;
    ofstream fo("KC.OUT");
    fo << m;
    fo.close();
    return 0;
}
```

2. Bài 2. BÁN DƯA

Năm nay được mùa dưa, Tý theo bố mang rất nhiều quả dưa ra chợ bán. Có một ông thương gia đến chợ mua dưa, biết Tý là người thông minh, ông ta muốn đưa ra thử thách cho Tý. Ông yêu cầu Tý thực hiện theo đúng trình tự sau:

Bước 1: Tý phải chọn ra N viên đá từ đống đá có sẵn ở chợ.

Bước 2: Trong N viên đá được chọn, loại ra những viên đá có khối lượng giống nhau.

Bước 3: Ông ta sẽ mua những quả dưa có tổng khối lượng đúng bằng tổng khối lượng của những viên đá còn lại sau khi thực hiện bước 2.

Tý đã rất nhanh chóng thực hiện yêu cầu trên và bán được số dưa có tổng khối lượng lớn nhất.

Yêu cầu: Em hãy tìm ra cách Tý đã làm.

Input: Đọc ở file văn bản **DUA.INP** có cấu trúc như sau:

- Dòng thứ nhất chứa 2 số nguyên dương N và K ($N \le K$), mỗi số cách nhau một dấu cách.
- Dòng thứ hai chứa K số nguyên dương, mỗi số cách nhau một dấu cách. Mỗi số trong dãy số này biểu thị cho khối lượng của mỗi viên đá và mỗi số không vượt quá 10000.
- Số lượng các viên đá có khối lượng nhỏ không ít hơn số lượng các viên đá có khối lượng lớn hơn.

Output: Ghi ra file văn bản **DUA.OUT** một số nguyên m là khối lượng dưa lớn nhất mà bố của Tý bán được.

Ví dụ:

DUA.INP	DUA.OUT
8 14 1 7 9 6 4 3 12 6 7 3 4 1 1 3	41

Giải thích: Tý chọn 8 viên đá có khối lượng là 1; 7; 9; 6; 4; 3; 12; 1 sau đó bỏ đi 2 viên có khối lượng bằng 1 thì tổng khối lượng còn lại là 41.

Giới hạn: 80% test ứng $K \le 10^4$; 20% test ứng với $10^4 < K \le 10^5$.

2.1. Tóm tắt đề bài:

- Trong một đống đá có **K** viên đá.
- Hãy lấy ra **<u>đủ</u> N** viên đá trong **K** viên đá **(N ≤ K)**.
- **Nếu** trong **N** viên đá được lấy ra mà có **từ 2 viên đá trở lên có khối lượng bằng nhau, thì** phải bỏ đi tất cả số viên đá có khối lượng bằng nhau đó.

=> Hỏi làm cách nào để Tý có thể lấy ra được các viên đá theo yêu cầu, sao cho tổng khối lượng thu được là lớn nhất?

2.2. Ý tưởng giải:

- **Bước 1.** Xếp các viên đá có khối lượng bằng nhau vào một nhóm. Sau khi sắp xếp xong Tý sẽ có được **n1** nhóm đá.
- **Bước 2.** Sắp xếp **n1** nhóm đá đó lại theo thứ tự giảm dần (Từ lớn đến nhỏ, từ trái qua phải).
- **Bước 3.** Trong **n1** nhóm đá đã sắp xếp, Tý sẽ lấy theo thứ tự từ trái qua phải, mỗi nhóm đá lấy **1** viên cho đến khi nào đủ **N** viên thì dừng.
- **Bước 4.**

  + Nếu số lượng các nhóm đá hiện có nhỏ hơn **N** và số lượng viên đá có trong nhóm cuối cùng (Nhóm đá mà các viên đá có khối lượng nhỏ nhất) **<u>nhiều hơn</u> 1 viên** thì Tý không lấy viên đá nào trong nhóm này.

  + Nếu số lượng các nhóm đá hiện có nhỏ hơn **N** và số lượng viên đá có trong nhóm cuối cùng (Nhóm đá mà các viên đá có khối lượng nhỏ nhất) **<u>chỉ còn duy nhất</u> 1 viên** thì Tý lấy luôn viên đá trong nhóm này.

=> Kết quả thu được chính là số lượng viên đá lấy được có khối lượng lớn nhất.

2.3. Mô tả thuật toán:

- **Bước 1. Đọc dữ liệu đầu vào** từ file DUA.INP:
  + Hai số nguyên N, K.
  + Một dãy số nguyên **a**.
- **Bước 2. Tiền xử lý mảng a:**
  + Sắp xếp mảng **a** theo thứ tự **giảm dần**.
- **Bước 3. Khởi tạo**:
  + Tạo mảng rỗng `b` để lưu các phần tử **không trùng nhau** của **a**.
  + Thêm phần tử đầu tiên của **a** vào `b`.
  + Tạo biến đếm `dem = 1` (số lượng phần tử không trùng nhau của `a`).
- **Bước 4. Duyệt mảng a:**
  + Với mỗi phần tử **a[i]**, nếu nó **khác phần tử đứng liền kề sau nó** và số lượng `dem` ít hơn **N** thì:
    - Tăng `dem` thêm **1** đơn vị.
    - Thêm **a[i + 1]** vào `b`.
  + Đếm số lần xuất hiện của **phần tử cuối cùng của mảng b trong mảng a**.
  + Nếu số lượng phần tử trong mảng `b` nhỏ hơn **N** và số lần xuất hiện của **phần tử cuối cùng của mảng b trong mảng a nhiều hơn 1** lần thì loại bỏ phần tử cuối trong `b`.

- **Bước 5. Xử lý sau vòng lặp:**
  + Tính tổng tất cả phần tử trong `b` → được giá trị `m`.
- **Bước 6. Ghi kết quả** ra file `DUA.OUT`.

2.4. Code tham khảo (Python và C++):

2.4.1. Code Python:

```
# Mo file de doc
fi = open('DUA.INP', 'r')
# Nhap gia tri N, K
N, K = fi.readline().split()
N, K = int(N), int(K)
# Nhap mang a
a = fi.read().split()
a = list(map(int, a))
# Dong file
fi.close()
# Sap xep mang a giam dan
a.sort(reverse=True)
# Tao mang rong b de luu cac phan tu khong trung nhau cua a
b = []
# Them phan tu lon nhat cua mang a vao mang b
b.append(a[0])
# Khoi tao bien dem de dem so luong phan tu khong trung nhau trong mang a
dem = 1
# Duyet mang a de tim cac phan tu khong trung nhau roi them vao mang b
for i in range(0, K - 1):
    # Voi moi phan tu a[i], neu no khac voi phan tu dung lien ke sau no va dem < N
thi:
    if (a[i] != a[i + 1]) and (dem < N):
        # Tang bien dem them 1 don vi
        dem = dem + 1
        # Them a[i + 1] vao b
        b.append(a[i + 1])
# Dem so lan xuat hien cua phan tu cuoi cung cua mang b trong mang a
solan = a.count(b[len(b) - 1])
# Neu so luong phan tu trong mang b nho hon N va so lan xuat hien cua phan tu cuoi cung
cua mang b trong mang a nhieu hon 1 lan thi loai bo di phan tu cuoi trong b
if dem < N and solan > 1: b.pop()
# Tinh tong cac phan tu trong b
m = sum(b)
# Mo file de ghi
fo = open('DUA.OUT', 'w')
# Ghi ket qua vao file
fo.write(f'{m}')
# Dong file
fo.close()
```

2.4.2. Code C++:

```
#include <iostream>
#include <bits/stdc++.h>
#include <fstream> // Khai bao cai nay de su dung ifstream va ofstream
#include <vector> // Khai bao cai nay de su dung vector
using namespace std;
int main()
{
    // Mo file de doc
    ifstream fi("DUA.INP");
    // Nhap gia tri N, K
    int N, K;
    fi >> N >> K;
    // Nhap mang a
    vector <int> a(K);
    for (int i = 0; i < K; i++) fi >> a[i];
    // Dong file
    fi.close();
    // Sap xep mang a giam dan
    sort(a.begin(), a.end(), greater<int>());
    // Tao mang rong b de luu cac phan tu khong trung nhau cua a
    vector <int> b;
    // Them phan tu lon nhat cua mang a vao mang b
    b.push_back(a[0]);
    // Khoi tao bien dem de dem so luong phan tu khong trung nhau trong mang a
    int dem = 1;
    // Duyet mang a de tim cac phan tu khong trung nhau roi them vao mang b
    for (int i = 0; i < K - 1; i++)
    {
        // Voi moi phan tu a[i], neu no khac voi phan tu dung lien ke sau no va dem <
N thi:
        if (a[i] != a[i + 1] && dem < N)
        {
            // Tang bien dem them 1 don vi
            dem = dem + 1;
            // Them a[i + 1] vao b
            b.push_back(a[i + 1]);
        }
    }
    // Dem so lan xuat hien cua phan tu cuoi cung cua mang b trong mang a
    int solan = count(a.begin(), a.end(), b[b.size() - 1]);
    // Neu so luong phan tu trong mang b nho hon N va so lan xuat hien cua phan tu
cuoi cung cua mang b trong mang a nhieu hon 1 lan thi loai bo di phan tu cuoi trong b
    if (dem < N && solan > 1) b.pop_back();
    // Tinh tong cac phan tu trong b
    long long m = 0;
    for (int i = 0; i < b.size(); i++) m = m + b[i];
```

```
    // Mo file de ghi
    ofstream fo("DUA.OUT");
    // Ghi ket qua vao file
    fo << m;
    // Dong file
    fo.close();
    return 0;
}
```

3. Bài 3. SỐ CHỮ SỐ

Người ta viết liên tiếp các số chính phương 1^2; 2^2; 3^2; 4^2; 5^2; …; k^2 thì được một số tự nhiên x = 149162536…

Yêu cầu: Xác định số lượng các chữ số của x.

Input: Đọc từ file văn bản **SCS.INP** một số nguyên dương k.

Output: Ghi ra file văn bản **SCS.OUT** số nguyên m theo yêu cầu.

Ví dụ:

SCS.INP	SCS.OUT
9	15

Giải thích: Với k = 9 ta có x = 149162536496481 là số tự nhiên có 15 chữ số.

Giới hạn: 30% test ứng với $k \le 10^6$; 40% test ứng với $10^6 < k \le 10^9$; 30% test ứng với $10^9 < k \le 10^{12}$.

3.1. Tóm tắt đề bài:

Nhập vào 1 số nguyên dương **k**. Yêu cầu tính số lượng chữ số của từng số có trong dãy 1^2; 2^2; 3^2; 4^2; 5^2; …; k^2 rồi cộng dồn số lượng các chữ số của từng số lại với nhau để được kết quả **m**.

3.2. Ý tưởng giải:

- Vì ràng buộc của đề bài là **k <= 10^{12}**, nên ta không thể dùng vòng lặp **for** để tính kết quả của từng số chính phương, rồi cộng độ dài của từng số chính phương lại với nhau để ra kết quả **m** được. Cách tối ưu hơn là dùng công thức để tính:
- Ta muốn biết tổng số chữ số của dãy **1^2; 2^2; 3^2; 4^2; 5^2; ..., k^2** khi nối lại. Thay vì tính từng **i^2**, ta nhóm các số **i** theo số chữ số của **i^2**:

  + Một số **i^2** có **d** chữ số khi nó nằm trong khoảng: **$10^{d-1} \le i^2 \le 10^d - 1$**

  + Lấy căn bậc 2: **sqrt(10^{d-1}) ≤ i < sqrt(10^d - 1)** => Nghĩa là: Tất cả các số **i** trong đoạn này đều có bình phương bằng **d** chữ số.

- Hướng giải:

  + Gọi:

    dau = ceil(sqrt(10^(d-1))) => **ceil:** Làm tròn lên đến số nguyên gần nhất

    cuoi = floor(sqrt(10^d - 1)) => **floor:** Làm tròn xuống số nguyên gần nhất

  + Số lượng số **i** trong đoạn là: **soluong = cuoi - dau + 1** => Với mỗi **d** sẽ đóng góp **soluong * d** chữ số

  + Lặp qua từng giá trị của **d**, cộng dồn tất cả **soluong * d** sẽ ra kết quả **m**. Nếu **k < cuoi** thì chỉ lấy đến **k** (Vì **k** luôn nhỏ hơn hoặc bằng **cuoi**)

3.3. Mô tả thuật toán:

- **Bước 1. Nhập dữ liệu**:
  + Đọc số nguyên dương k.
- **Bước 2. Khởi tạo kết quả**:
  + Gán **m = 0**.
  + Biến **m** sẽ giữ tổng số chữ số của tất cả các bình phương $\mathbf{i^2}$ với $\mathbf{1 \le i \le k}$.
- **Bước 3. Duyệt số chữ số d của các bình phương**:
  + Vì theo đề bài cho $\mathbf{k \le 10^{12}}$ => $\mathbf{k^2 = 24}$, vậy **24** là số lượng chữ số lớn nhất nên ta chỉ duyệt **d** từ **1** đến **24**.
  + Với mỗi **d**, ta tìm mọi số nguyên **i** sao cho $\mathbf{i^2}$ có đúng **d** chữ số.
- **Bước 4.** Tính cận dưới **(dau)** và cận trên **(cuoi)** của **i** sao cho $\mathbf{i^2}$ có **d** chữ số:
  + Điều kiện: $\mathbf{10^{d-1} \le i^2 \le 10^d - 1}$
  + Do đó:
    - **dau = math.isqrt(10**(d-1))**: Là **i** nhỏ nhất sao cho $\mathbf{i^2}$ có **d** chữ số.
    - **cuoi = math.isqrt(10**d - 1)**: Là **i** lớn nhất sao cho $\mathbf{i^2}$ có **d** chữ số.
- **Bước 5. Giới hạn theo k**:
  + Nếu **dau > k** thì không còn **i** nào thỏa ở các **d** tiếp theo -> **dừng vòng lặp (break)**.
  + Nếu **cuoi > k** thì đặt **cuoi = k** (vì ta chỉ xét **i ≤ k**).
- **Bước 6. Nếu khoảng hợp lệ (dau ≤ cuoi):**
  + Số lượng số **i** trong khoảng là: **soluong = cuoi - dau + 1**.
  + Tổng số chữ số do những **i** này đóng góp là: **soluong * d**.
  + Cộng dồn vào **m**: **m = m + soluong * d**.
- **Bước 7.** Tiếp tục với các d tiếp theo (Bước 3) cho đến khi **d = 24** hoặc bị **break** vì **dau > k**.
- **Bước 8. In Kết quả:**
  + Sau khi thực hiện vòng lặp **for** xong thì **m** chính là tổng số chữ số của tất cả $\mathbf{i^2}$ với $\mathbf{1 \le i \le k}$.
  + Ghi **m** ra file **.OUT**

3.4. Code tham khảo (Python và C++):

3.4.1. Code Python:

```
# Khai bao thu vien nay de co the tinh duoc sqrt
import math
# Mo file de doc
fi = open('SCS.INP', 'r')
# Nhap gia tri k
k = fi.read()
k = int(k)
# Dong file
fi.close()
# Khoi tao m = 0 de luu lai so luong chu so cua day
m = 0
```

```
# Vi de bai yeu cau k lon nhat la k^12 => So luong chu so d nhieu nhat co the co la
(k^12)^2 = k^24, nen so luong chu so d nhieu nhat chi co the la 24 (d = 1 .. 24)
for d in range(1, 25):
    # Tinh can duoi cua i, lam tron len den so nguyen gan nhat
    dau = math.isqrt(10**(d-1))
    if dau * dau < 10**(d-1): dau = dau + 1
    # Tinh can tren cua i, lam tron xuong so nguyen gan nhat
    cuoi = math.isqrt(10**d - 1)
    # Neu dau > k thi khong co so de tinh
    if dau > k: break
    # Neu cuoi > k thi chi lay cuoi = k, vi k luon nho hon hoac bang cuoi
    if cuoi > k: cuoi = k
    # Neu dau <= cuoi thi tinh so luong so i trong doan la:
    if dau <= cuoi:
        # Cong thuc tinh so luong chu so trong khoang i co d chu so
        soluong = cuoi - dau + 1
        # Cong don tat ca soluong * d de ra ket qua m
        m = m + soluong * d
# Mo file de ghi
fo = open('SCS.OUT', 'w')
# Ghi ket qua vao file
fo.write(f'{m}')
# Dong file
fo.close()
```

3.4.2. Code C++:

```
#include <iostream>
#include <bits/stdc++.h> // Khai bao cai nay de su dung duoc day du thu vien
toan hoc
#include <fstream> // Khai bao cai nay de su dung ifstream va ofstream
using namespace std;
int main()
{
    // Mo file de doc
    ifstream fi("SCS.INP");
    // Nhap gia tri k
    long long k;
    fi >> k;
    // Dong file
    fi.close();
    // Khoi tao m = 0 de luu lai so luong chu so cua day
    long long m = 0;
    // Vi de bai yeu cau k lon nhat la k^12 => So luong chu so d nhieu nhat co the co
la (k^12)^2 = k^24, nen so luong chu so d nhieu nhat chi co the la 24 (d = 1 .. 24)
    for (int d = 1; d < 25; d++)
```

```
    {
        // Tinh can duoi cua i, lam tron len den so nguyen gan nhat
        long long dau = sqrtl(pow(10.0L, d - 1));
        if (dau * dau < (long long)pow(10, d - 1)) dau = dau + 1;
        // Tinh can tren cua i, lam tron xuong so nguyen gan nhat
        long long cuoi = sqrtl(powl(10, d) - 1);
        // Neu dau > k thi khong co so de tinh
        if (dau > k) break;
        // Neu cuoi > k thi chi lay cuoi = k, vi k luon nho hon hoac bang cuoi
        if (cuoi > k) cuoi = k;
        // Neu dau <= cuoi thi tinh so luong so i trong doan la:
        if (dau <= cuoi)
        {
            // Cong thuc tinh so luong chu so trong khoang i co d chu so
            long long soluong = cuoi - dau + 1;
            // Cong don tat ca soluong * d de ra ket qua m
            m = m + soluong * d;
        }
    }
    // Mo file de ghi
    ofstream fo("SCS.OUT");
    // Ghi ket qua vao file
    fo << m;
    // Dong file
    fo.close();
    return 0;
}
```

4. Các bộ Test:

Mỗi câu của đề thi đều có đủ 10 bộ Test, với các giá trị đầu vào tương ứng với tỉ lệ (%) của các ràng buộc (Substack) mà đề bài yêu cầu. Những bộ Test này đã được gửi ở tệp tin đính kèm cùng với tài liệu này.

5. Kết quả chấm thử bằng phần mềm Themis 1.9, giới hạn thời gian chạy của mỗi bộ Test là 01 giây:

Tên bài	Điểm tối đa	Điểm chấm thử bằng Themis 1.9	
		Python	C++
Bài 1. KHOẢNG CÁCH (KC.*)	4.0 điểm	4.0 điểm (100%)	4.0 điểm (100%)
Bài 2. BÁN DƯA (DUA.*)	3.0 điểm	3.0 điểm (100%)	3.0 điểm (100%)
Bài 3. SỐ CHỮ SỐ (SCS.*)	3.0 điểm	3.0 điểm (100%)	2.1 điểm (70%)
Tổng số điểm đạt được:		**10.0 điểm (100%)**	**9.1 điểm (91%)**

ĐỀ SỐ 6
(Học sinh giỏi 9 – Đà Nẵng – Năm học: 2021 – 2022)

SỞ GIÁO DỤC VÀ ĐÀO TẠO
THÀNH PHỐ ĐÀ NẴNG

KÌ THI CHỌN HỌC SINH GIỎI LỚP 9 CẤP THÀNH PHỐ
NĂM HỌC 2021 - 2022

ĐỀ CHÍNH THỨC

MÔN: TIN HỌC
Thời gian làm bài: 150 phút *(không kể thời gian giao đề)*
(Đề thi có 2 trang)

TỔNG QUAN

	Tên bài	File chương trình	File dữ liệu vào	File dữ liệu ra
Câu 1	Số nguyên tố	SNT.*	Nhập từ bàn phím	In ra màn hình
Câu 2	Đếm kí tự	DEMKT.*	Nhập từ bàn phím	In ra màn hình
Câu 3	Tam giác	TAMGIAC.*	TAMGIAC.INP	TAMGIAC.OUT
Câu 4	Bội đặc biệt	BOI.*	BOI.INP	BOI.OUT

*Phần mở rộng * là pas, c, cpp hay py tùy theo ngôn ngữ và môi trường lập trình.*

Câu 1. SNT - Số nguyên tố *(2 điểm)*

Nhập từ bàn phím một số nguyên dương a. Hãy tìm và in ra màn hình số nguyên dương b nhỏ nhất sao cho tổng a + b là một số nguyên tố.

Ví dụ:

Nhập từ bàn phím	In ra màn hình
5	2

Giải thích: *Số 2 là số nguyên dương nhỏ nhất thỏa mãn 5 + 2 = 7 là một số nguyên tố.*

Giới hạn: 70% test ứng với $0 < a < 106$; 30% test ứng với $106 \le a \le 109$.

Câu 2. DEMKT- Đếm kí tự *(3 điểm)*

Nhập từ bàn phím một xâu kí tự S. Hãy in ra màn hình số lượng các kí tự chữ cái A có trong xâu S.

Ví dụ:

Nhập từ bàn phím	In ra màn hình
mmaeaAbA	2

Giới hạn: Xâu kí tự S có không quá 255 kí tự.

HSG Tin học lớp 9 năm học 2021 - 2022 *Trang 1/2*

Câu 3. TAMGIAC - Tam giác *(2 điểm)*

Cho trước 2 số nguyên dương a và b.

Yêu cầu: Xác định có bao nhiêu số nguyên dương c sao cho 3 số a; b; c là độ dài 3 cạnh của một tam giác.

Dữ liệu vào: Đọc từ file văn bản **TAMGIAC.INP** hai số a và b trên cùng một dòng, mỗi số cách nhau một dấu cách.

Dữ liệu ra: Ghi ra file văn bản **TAMGIAC.OUT** số m là số các số c thỏa mãn yêu cầu.

Ví dụ:

TAMGIAC.INP	TAMGIAC.OUT
3 4	5

Giải thích: *Có 5 số c thỏa mãn là: 2; 3; 4; 5; 6.*

Giới hạn: $0 < a; b < 10^9$.

Câu 4. BOI - Bội đặc biệt *(3 điểm)*

Cho trước một số nguyên dương P không chia hết cho 2 và 5. Số X được gọi là một bội đặc biệt của P nếu số X chia hết cho P và X chỉ chứa toàn chữ số 9.

Ví dụ: Với P = 7 thì X = 999999 là một bội đặc biệt của số P.

Yêu cầu: Cho trước hai nguyên dương P và N. Hỏi có bao nhiêu số X là bội đặc biệt của P và X có số chữ số không vượt quá N.

Dữ liệu vào: Đọc từ file văn bản **BOI.INP** hai số P và N trên cùng một dòng, mỗi số cách nhau một dấu cách.

Dữ liệu ra: Ghi ra file văn bản **BOI.OUT** số m là số lượng các số X thỏa mãn yêu cầu.

Ví dụ:

BOI.INP	BOI.OUT
7 7	1

Giải thích: *Với P = 7 và N = 7 ta có duy nhất một số X = 999999 thỏa mãn.*

Giới hạn: 30% test tương ứng với $P < 10^2$ và $N \le 10$;
40% test tương ứng với $10^2 \le P < 10^4$ và $N \le 160$;
30% test tương ứng với $10^4 \le P < 10^6$ và $N \le 10^{18}$.

---Hết---

BÀI GIẢI ĐỀ SỐ 6
(Học sinh giỏi 9 – Đà Nẵng – Năm học: 2021 – 2022)

1. Câu 1. SNT - Số nguyên tố *(2 điểm)*

Câu 1. SNT - Số nguyên tố *(2 điểm)*

Nhập từ bàn phím một số nguyên dương a. Hãy tìm và in ra màn hình số nguyên dương b nhỏ nhất sao cho tổng a + b là một số nguyên tố.

Ví dụ:

Nhập từ bàn phím	In ra màn hình
5	2

Giải thích: *Số 2 là số nguyên dương nhỏ nhất thỏa mãn 5 + 2 = 7 là một số nguyên tố.*

Giới hạn: 70% test ứng với $0 < a < 106$; 30% test ứng với $106 \le a \le 109$.

1.1. Tóm tắt đề bài:

Nhập vào số nguyên dương a. Tìm giá trị b nhỏ nhất sao cho a + b là số nguyên tố.

1.2. Ý tưởng giải:

- Đọc số nguyên **a** từ file **SNT.INP**.
- Khởi tạo **b = 1**.
- Kiểm tra **a + b** có phải số nguyên tố không:
  + Nếu chưa, tăng **b** lên **1** và kiểm tra lại.
  + Nếu rồi, dừng vòng lặp.
- Ghi **b** vào file **SNT.OUT**.

1.3. Mô tả thuật toán:

- **Bước 1.** Khai báo hàm kiểm tra số nguyên tố
- **Bước 2.** Đọc số nguyên **a** từ file **SNT.INP**.
- **Bước 3.** Khởi tạo **b = 1**.
- **Bước 4.** Kiểm tra **a + b** có phải số nguyên tố không:
  + Nếu chưa, tăng **b** lên **1** và kiểm tra lại.
  + Nếu rồi, dừng vòng lặp.
- **Bước 5.** Ghi **b** vào file **SNT.OUT**.

1.4. Code tham khảo (Python và C++):

1.4.1. Code Python:

```
import math # Thư viện để dùng hàm căn bậc hai
# ===== HÀM KIỂM TRA NGUYÊN TỐ =====
# Tạo hàm kiểm tra số nguyên tố
def nguyento(x):
    # Nếu x <= 1 thì không phải số nguyên tố
    if x <= 1: return False
    # Nếu x là 2 hoặc 3 thì là số nguyên tố
    if x == 2 or x == 3: return True
```

```
    # Nếu x chia hết cho 2 hoặc 3 thì không phải số nguyên tố
    if x % 2 == 0 or x % 3 == 0: return False
    # Bắt đầu từ 5 (vì số nguyên tố > 3 có dạng 6k ± 1)
    i = 5
    # Kiểm tra đến căn bậc hai của x
    while i <= math.isqrt(x):
        # Nếu chia hết cho i hoặc i+2 thì không phải số nguyên tố
        if x % i == 0 or x % (i + 2) == 0:
            return False
        # Mỗi bước tăng 6 (kiểm tra cặp 6k-1 và 6k+1)
        i = i + 6
    # Nếu không chia hết cho số nào, x là số nguyên tố
    return True
# ===== CHƯƠNG TRÌNH CHÍNH =====
fi = open('SNT.INP', 'r')    # Mở file để doc
a = fi.read()                # Nhập a
a = int(a)                   # Chuyển chuỗi đọc được thành số nguyên
fi.close()                   # Đóng file sau khi đọc xong
b = 1                                # Gán giá trị ban đầu cho b là 1
while nguyento(a + b) == False:      # Trong khi (a + b) chưa phải là số
nguyên tố
    b = b + 1                        # Thì tăng b lên 1 rồi kiểm tra lại
fo = open('SNT.OUT', 'w')            # Mở file "SNT.OUT" để ghi kết quả
fo.write(f'{b}')                     # Ghi giá trị b (kết quả tìm được) vào
file
fo.close()                           # Đóng file sau khi ghi xong
```

1.4.2. Code C++:

```
#include <iostream>
#include <bits/stdc++.h>     // Thu vien tong hop trong C++ (bao gom math,
iostream, fstream, v.v.)
#include <fstream>           // Khai bao cai nay de su dung ifstream va
ofstream
using namespace std;
// ===== HAM KIEM TRA NGUYEN TO =====
bool nguyento(long long x)
{
    // Neu x <= 1 thi khong phai so nguyen to
    if (x <= 1) return false;
    // Neu x la 2 hoac 3 thi la so nguyen to
    if (x == 2 || x == 3) return true;
    // Neu x chia het cho 2 hoac 3 thi khong phai so nguyen to
    if (x % 2 == 0 || x % 3 == 0) return false;
    // Bat dau kiem tra tu 5 (vi so nguyen to > 3 co dang 6k ± 1)
    long long i = 5;
    // Kiem tra den can bac hai cua x
    while (i <= sqrt(x))
    {
```

```
        // Neu x chia het cho i hoac i+2 thi khong phai so nguyen to
        if (x % i == 0 || x % (i + 2) == 0) return false;
        // Moi buoc tang 6 (kiem tra cap 6k-1 va 6k+1)
        i = i + 6;
    }
    // Neu khong chia het cho so nao, x la so nguyen to
    return true;
}
// ===== CHUONG TRINH CHINH =====
int main()
{
    ifstream fi("SNT.INP");                 // Mo file de doc
    int a;
    fi >> a;                                // Nhap a
    fi.close();                             // Dong file sau khi doc xong
    int b = 1;                              // Gan gia tri ban dau cho b la 1
    while (nguyento(a + b) == false)        // Trong khi (a + b) chua phai la
so nguyen to
        b = b + 1;                          // Thi tang b len 1 roi kiem tra
lai
    ofstream fo("SNT.OUT");                 // Mo file "SNT.OUT" de ghi ket qua
    fo << b;                                // Ghi gia trị b (ket qua tim duoc)
vao file
    fo.close();                             // Dong file sau khi ghi xong
    return 0;
}
```

2. Câu 2. DEMKT - Đếm kí tự *(3 điểm)*

Câu 2. DEMKT- Đếm kí tự *(3 điểm)*

Nhập từ bàn phím một xâu kí tự S. Hãy in ra màn hình số lượng các kí tự chữ cái A có trong xâu S.

Ví dụ:

Nhập từ bàn phím	In ra màn hình
mmaeaAbA	2

Giới hạn: Xâu kí tự S có không quá 255 kí tự.

2.1. Tóm tắt đề bài:

Nhập vào xâu kí tự **S**. Hãy đếm xem trong xâu **S** vừa nhập có bao nhiêu kí tự **A**

2.2. Ý tưởng giải:

Duyệt từ đầu đến cuối xâu S, khi nào gặp kí tự **A** thì tăng biến đếm (**dem**) lên **1** đơn vị

2.3. Mô tả thuật toán:

- **Bước 1.** Nhập vào xâu kí tự **S**
- **Bước 2.** Khởi tạo biến **dem = 0**
- **Bước 3.** Duyệt từ đầu đến cuối xâu **S**, khi nào gặp kí tự **A** thì tăng biến **dem** lên **1** đơn vị **(dem = dem + 1)**
- **Bước 4.** In kết quả là biến **dem**

2.4. Code tham khảo (Python và C++):

2.4.1. Code Python:

```
fi = open('DEMKT.INP', 'r')          # Mở file để đọc
S = fi.read()                        # Đọc toàn bộ nội dung file vào chuỗi S
fi.close()                           # Đóng file sau khi đọc xong
dem = 0                              # Biến đếm số ký tự 'A'
for i in range(0, len(S)):           # Duyệt từng ký tự trong chuỗi
    if S[i] == 'A': dem = dem + 1    # Nếu ký tự là 'A' thì tăng biến đếm lên
1
fo = open('DEMKT.OUT', 'w')          # Mở file "DEMK.OUT" để ghi kết quả
fo.write(f'{dem}')                   # Ghi kết quả (số lượng ký tự 'A') vào
file
fo.close()                           # Đóng file sau khi ghi xong
```

2.4.2. Code C++:

```
#include <iostream>
#include <bits/stdc++.h>          // Thu vien tong hop
#include <fstream>                // Khai bao cai nay de su dung ifstream va
ofstream
using namespace std;
int main()
{
    ifstream fi("DEMKT.INP");     // Mo file de doc
    string S;
    getline(fi, S);               // Doc toan bo noi dung file vao chuoi S
    fi.close();                   // Dong file sau khi doc xong
    int dem = 0;                  // Bien demm so ky tu 'A'
    for (int i = 0; i < S.length(); i++)     // Duyet tung ky tu trong
chuoi
    {
        if (S[i] == 'A') dem = dem + 1;      // Neu ky tu la 'A' thi tang
bien dem len 1
    }
    ofstream fo("DEMKT.OUT");     // Mo file "DEMK.OUT" de ghi ket qua
    fo << dem;                    // Ghi ket qua (so luong ky tu 'A') vao file
    fo.close();                   // Dong file sau khi ghi xong
    return 0;
}
```

3. Câu 3. TAMGIAC - Tam giác *(2 điểm)*

Câu 3. TAMGIAC - Tam giác *(2 điểm)*

Cho trước 2 số nguyên dương a và b.

Yêu cầu: Xác định có bao nhiêu số nguyên dương c sao cho 3 số a; b; c là độ dài 3 cạnh của một tam giác.

Dữ liệu vào: Đọc từ file văn bản **TAMGIAC.INP** hai số a và b trên cùng một dòng, mỗi số cách nhau một dấu cách.

Dữ liệu ra: Ghi ra file văn bản **TAMGIAC.OUT** số m là số các số c thỏa mãn yêu cầu.

Ví dụ:

TAMGIAC.INP	TAMGIAC.OUT
3 4	5

Giải thích: *Có 5 số c thỏa mãn là: 2; 3; 4; 5; 6.*

Giới hạn: $0 < a; b < 10^9$.

3.1. Tóm tắt đề bài:

Cho hai cạnh **a** và **b**, hãy đếm xem có bao nhiêu số nguyên dương **c** sao cho ba cạnh **a, b, c** có thể tạo thành một tam giác.

3.2. Ý tưởng giải: Xét điều kiện để 3 cạnh tạo thành tam giác

- Với 3 cạnh a, b, c:

$$|a-b| < c < a+b$$

- Vì c là số nguyên dương nên ta lấy:

$$c \in [|a-b|+1, a+b-1]$$

- Số lượng các giá trị c thỏa mãn là:

$$\text{count} = (a+b-1) - (|a-b|+1) + 1 = a+b-|a-b|-1$$

3.3. Mô tả thuật toán:

- **Bước 1.** Nhập dữ liệu từ file **TAMGIAC.INP**
- **Bước 2.** Sử dụng công thức sau để tính số lượng số **c**:

count = a + b - abs(a - b) – 1

- **Bước 3.** Ghi **count** ra file **TAMGIAC.OUT**

3.4. Code tham khảo (Python và C++):

3.4.1. Code Python:

```
fi = open('TAMGIAC.INP', 'r')        # Mo file de doc
a, b = fi.read().split()             # Nhap a, b
a, b = int(a), int (b)               # Chuyen a, b sang so nguyen
fi.close()                           # Dong file sau khi doc
count = a + b - abs(a - b) - 1       # Cong thuc tinh so luong so c
```

```
fo = open('TAMGIAC.OUT', 'w')        # Mo file de ghi
fo.write(f'{count}')                 # Ghi ket qua ra file
fo.close()                           # Dong file sau khi ghi xong
```

3.4.2. Code C++:

```
#include <iostream>
#include <bits/stdc++.h>     // Thu vien tong hop
#include <fstream>           // Khai bao cai nay de su dung ifstream va ofstream
using namespace std;
int main()
{
    ifstream fi("TAMGIAC.INP"); // Mo file de doc
    long long a, b;
    fi >> a >> b;               // Nhap a, b
    fi.close();                 // Dong file sau khi doc
    long long count = a + b - abs(a - b) - 1;    // Cong thuc tinh so
luong so c
    ofstream fo("TAMGIAC.OUT"); // Mo file de ghi
    fo << count;                // Ghi ket qua ra file
    fo.close();                 // Dong file sau khi ghi xong
    return 0;
}
```

4. Câu 4. BOI - Bội đặc biệt *(3 điểm)*

Câu 4. BOI - Bội đặc biệt *(3 điểm)*

Cho trước một số nguyên dương P không chia hết cho 2 và 5. Số X được gọi là một bội đặc biệt của P nếu số X chia hết cho P và X chỉ chứa toàn chữ số 9.

Ví dụ: Với P = 7 thì X = 999999 là một bội đặc biệt của số P.

Yêu cầu: Cho trước hai nguyên dương P và N. Hỏi có bao nhiêu số X là bội đặc biệt của P và X có số chữ số không vượt quá N.

Dữ liệu vào: Đọc từ file văn bản **BOI.INP** hai số P và N trên cùng một dòng, mỗi số cách nhau một dấu cách.

Dữ liệu ra: Ghi ra file văn bản **BOI.OUT** số m là số lượng các số X thỏa mãn yêu cầu.

Ví dụ:

BOI.INP	BOI.OUT
7 7	1

Giải thích: *Với P = 7 và N = 7 ta có duy nhất một số X = 999999 thỏa mãn.*

Giới hạn: 30% test tương ứng với $P < 10^2$ và $N \le 10$;
40% test tương ứng với $10^2 \le P < 10^4$ và $N \le 160$;
30% test tương ứng với $10^4 \le P < 10^6$ và $N \le 10^{18}$.

4.1. Tóm tắt đề bài:

- **Ta có:**
  + Một số nguyên dương **P không chia hết cho 2 và 5**
  + Một số **X** được gọi là **bội đặc biệt của P** nếu:
    ▪ **X** chia hết cho **P**
    ▪ **X** chỉ chứa các số **9**
- **Yêu cầu:**
  + Đếm số lượng **X** là bội đặc biệt của **P**, **số chữ số của X ≤ N**

4.2. Ý tưởng giải: Ta không thể duyệt hết các số như 9, 99, 999, … vì có thể lên đến 10^{18} chữ số. Thay vào đó, ta sử dụng tính chất chia dư để xử lí từng bước.

- Bắt đầu: Đặt **SỐ DƯ = 0**
- Lặp lại **N** lần:
  + Mỗi lần lặp tương ứng với việc thêm một chữ số **9** vào cuối số cũ. Khi đó: **SỐ DƯ = (SỐ DƯ * 10 + 9) % P**
  + Công thức trên có nghĩa là: *"Nhân số dư cũ với 10 (Vì thêm 1 chữ số), cộng 9 (Chữ số mới), rồi chia lấy dư cho P"*.
- **Đếm số lần mà SỐ DƯ = 0**, vì khi đó số tương ứng vừa xét sẽ **chia hết cho P**. **Số lần đếm được** chính là kết quả mà đề bài yêu cầu.

4.3. Mô tả thuật toán:

- **Bước 1.** Đọc **P, N** từ file **BOI.INP**
- **Bước 2.** Khởi tạo **2** biến ban đầu: **dem = 0, sodu = 0**
- **Bước 3.** Cho biến chạy **i** chạy từ **1** đến **N**:
  + Tính số dư theo công thức: **sodu = (sodu * 10 + 9) % P**
  + **Đếm** số lần mà **sodu = 0**
- **Bước 4.** Xuất kết quả tính được (**dem**) ra file **BOI.OUT**

4.4. Code tham khảo (Python và C++):

4.4.1. Code Python:

```python
fi = open('BOI.INP', 'r')    # Mo file de doc
P, N = fi.read().split()     # Nhap P, N
P, N = int(P), int (N)       # Chuyen P, N sang so nguyen
fi.close()                   # Dong file sau khi doc
dem = 0                      # Khoi tao bien dem = 0
sodu = 0                     # Khoi tao bien sodu = 0
for i in range(1, N + 1):            # Lap lai N lan
    sodu = (sodu * 10 + 9) % P       # Ap dung cong thuc chia du
    if sodu == 0:                    # Neu chia het
        dem = dem + 1                # Tang bien dem
fo = open('BOI.OUT', 'w')    # Mo file de ghi
fo.write(f'{dem}')           # Ghi ket qua ra file
fo.close()                   # Dong file sau khi ghi xong
```

4.4.2. Code C++:

```
#include <iostream>          // Thu vien nhap xuat
#include <bits/stdc++.h>     // Thu vien tong hop
#include <fstream>           // Thu vien doc ghi file
using namespace std;
int main()
{
    ifstream fi("BOI.INP");      // Mo file de doc
    long long P, N;              // Khai bao bien P, N
    fi >> P >> N;                // Nhap P, N
    fi.close();                  // Dong file sau khi doc
    long long dem = 0;           // Khoi tao bien dem = 0
    long long sodu = 0;          // Khoi tao bien sodu = 0
    for (long long i = 1; i < N + 1; i++)    // Lap lai N lan
    {
        sodu = (sodu * 10 + 9) % P;          // Ap dung cong thuc chia du
        if (sodu == 0)                       // Neu chia het
            dem = dem + 1;                   // Tang bien dem
    }
    ofstream fo("BOI.OUT"); // Mo file de ghi
    fo << dem;              // Ghi ket qua ra file
    fo.close();             // Dong file sau khi ghi xong
    return 0;
}
```

5. Các bộ Test:

Mỗi câu của đề thi đều có đủ 10 bộ Test, với các giá trị đầu vào tương ứng với tỉ lệ (%) của các ràng buộc (Substack) mà đề bài yêu cầu. Những bộ Test này đã được gửi ở tệp tin đính kèm cùng với tài liệu này.

6. Kết quả chấm thử bằng phần mềm Themis 1.9, giới hạn thời gian chạy của mỗi bộ Test là 01 giây:

Tên bài	Điểm tối đa	Điểm chấm thử bằng Themis 1.9	
		Python	C++
Câu 1. Số nguyên tố (SNT.*)	2.0 điểm	2.0 điểm (100%)	2.0 điểm (100%)
Câu 2. Đếm kí tự (DEMKT.*)	3.0 điểm	3.0 điểm (100%)	3.0 điểm (100%)
Câu 3. Tam giác (TAMGIAC.*)	2.0 điểm	2.0 điểm (100%)	2.0 điểm (100%)
Câu 4. Bội đặc biệt (BOI.*)	3.0 điểm	2.1 điểm (70%)	2.1 điểm (70%)
Tổng số điểm đạt được:		**9.1 điểm (91%)**	**9.1 điểm (91%)**

ĐỀ SỐ 7
(Tin học trẻ Bảng B (THCS) – Đà Nẵng – Năm học: 2021 – 2022)

UBND THÀNH PHỐ ĐÀ NẴNG
HỘI THI TIN HỌC TRẺ CẤP THÀNH PHỐ
NĂM HỌC 2021-2022

ĐỀ THI BẢNG B – CẤP THCS
Thời gian: 150 phút (không kể thời gian giao đề)
(Đề thi có 3 trang)

ĐỀ CHÍNH THỨC

TỔNG QUAN

	Tên bài	File chương trình	File dữ liệu vào	File dữ liệu ra
Bài 1	Đếm kí tự	DEMKT.*	Bàn phím	Màn hình
Bài 2	Siêu đối xứng	DX.*	Bàn phím	Màn hình
Bài 3	Giả thuyết Goldbach	GOL.*	GOL.INP	GOL.OUT
Bài 4	Đường đi của Robot	ROBOT.*	ROBOT.INP	ROBOT.OUT

*Chú ý: Dấu * có thể là pas; c; cpp hoặc py tương ứng với ngôn ngữ lập trình là Free Pacal; C/C++ hoặc Python.*

Bài 1. Đếm kí tự *(3.0 điểm)*

Nhập vào từ bàn phím một xâu kí tự S (xâu S có không quá 255 kí tự). Hãy đếm và in ra màn hình số lượng kí tự chữ số 9 có trong xâu S.

Ví dụ:

Nhập từ bàn phím	In ra màn hình
CdqC9Cac8ac96c2369	3

Giải thích: Có 3 kí tự chữ số 9 trong xâu

Bài 2. Siêu đối xứng *(2.0 điểm)*

Một số nguyên dương được gọi là siêu đối xứng nếu tất cả các chữ số của nó giống nhau. Chẳng hạn số 777 hoặc 4444 là các số nguyên dương siêu đối xứng.

Nhập từ bàn phím một số nguyên dương x. Hãy tìm và in ra màn hình **số nguyên dương y nhỏ nhất** sao cho tổng $x + y$ là một số nguyên dương siêu đối xứng.

Ví dụ:

Nhập từ bàn phím	In ra màn hình
45	10

Giải thích: $45 + 10 = 55$
Giới hạn: Có 50% test ứng với $x \le 10^6$; 30% test ứng với $10^6 < x \le 10^9$; 20% test ứng với $10^9 \le x \le 10^{16}$

Bài 3. Giả thuyết Goldbach *(3.0 điểm)*

Giả thuyết Goldbach do nhà toán học người Đức Christian Goldbach (1690-1764) nêu ra vào năm 1742 trong một lá thư gửi tới nhà toán học Leonhard Euler, là một trong những bài toán lâu đời và nổi tiếng còn chưa giải được trong lý thuyết số nói riêng và toán học nói chung. Giả thuyết phỏng đoán rằng:

Mỗi số tự nhiên chẵn lớn hơn 2 có thể biểu diễn bằng tổng của hai số nguyên tố.

Giả thuyết đã được chỉ ra là đúng tới 4×10^{18}, nhưng hiện nay vẫn chưa được chứng minh hoàn toàn.

Bằng kiến thức tin học em hãy thực hiện thử thách sau:

Cho trước số x là số tự nhiên chẵn lớn hơn 2.

Yêu cầu: Hãy cho biết có bao nhiêu cách phân tích số x thành tổng của 2 số nguyên tố.

Lưu ý: Nếu phân tích được $x = a + b$ hay $x = b + a$ (trong đó a và b là các số nguyên tố) thì cũng chỉ được tính là một cách phân tích.

Input: Đọc ở file văn bản **GOL.INP** một số x là số tự nhiên chẵn lớn hơn 2.

Output: Ghi ra file văn bản **GOL.OUT** một số m theo yêu cầu.

Ví dụ:

GOL.INP	GOL.OUT
10	2

Giải thích: $10 = 3 + 7 = 5 + 5$

Giới hạn: Có 50% test ứng với $x \leq 10^3$; 30% test ứng với $10^3 < x \leq 10^4$; 20% test ứng với $10^4 < x \leq 10^6$

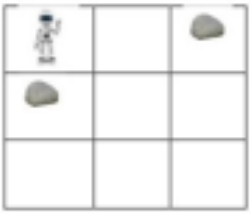
Bài 4. Đường đi của Robot *(2.0 điểm)*

Có một lưới ô vuông có kích thước $N \times N$ được đánh chỉ số hàng từ 1 đến N (theo chiều từ trên xuống dưới) và chỉ số cột từ 1 đến N (theo chiều từ trái sang phải). Mỗi ô trong lưới được xác định vị trí bởi một cặp số $(i; j)$ trong đó i là chỉ số hàng và j là chỉ số cột.

Tại ô $(1; 1)$ người ta đặt một con robot tự hành. Mỗi lần di chuyển robot chỉ đi sang phải một ô hoặc đi xuống dưới một ô. Trong lưới ô vuông này người ta đặt một viên đá vào một số ô để làm vật cản.

Yêu cầu: Hãy tính xem có bao nhiêu đường đi từ ô $(1; 1)$ đến ô $(N; N)$. Biết rằng robot không thể đi vào ô có vật cản và hai đường đi được gọi là khác nhau nếu có ít nhất một ô thuộc đường đi này nhưng không thuộc đường đi kia.

Vd: Xét lưới ô vuông kích thước 3x3 như hình vẽ sau:



Trong lưới ô vuông 3x3 này người ta đặt viên đá vào ô (1;3) và ô (2;1).
Với dữ kiện trên thì robot có tất cả 2 đường đi như sau:
(1;1) → (1;2) → (2;2) → (2;3) → (3;3)
(1;1) → (1;2) → (2;2) → (3;2) → (3;3)

Input: Đọc từ file văn bản **ROBOT.INP** có cấu trúc như sau:

- Dòng đầu tiên chứa 2 số nguyên dương N và M (mỗi số cách nhau 1 dấu cách; M < N).
- M dòng tiếp theo, mỗi dòng ghi 2 số nguyên dương i và j (mỗi số cách nhau một dấu cách) là chỉ số hàng và chỉ số cột của ô được đặt vào đó một viên đá là vật cản.

Output: Ghi ra file văn bản **ROBOT.OUT** một số k là số đường đi của robot từ ô $(1; 1)$ đến ô $(N; N)$.

Ví dụ:

ROBOT.INP	ROBOT.OUT
3 2 1 3 2 1	2

Giới hạn: 30% test ứng với $N \leq 10$; 40% test ứng với $10 < N \leq 30$; 30% test ứng với $30 < N \leq 100$;

-------*Hết*-------

BÀI GIẢI ĐỀ SỐ 7
(Tin học trẻ Bảng B (THCS) – Đà Nẵng – Năm học: 2021 – 2022)

1. Bài 1. Đếm kí tự *(3.0 điểm)*

Bài 1. Đếm kí tự *(3.0 điểm)*

Nhập vào từ bàn phím một xâu kí tự S (xâu S có không quá 255 kí tự). Hãy đếm và in ra màn hình số lượng kí tự chữ số 9 có trong xâu S.

Ví dụ:

Nhập từ bàn phím	In ra màn hình
CdqC9Cac8ac96c2369	3

Giải thích: Có 3 kí tự chữ số 9 trong xâu

1.1. Tóm tắt đề bài:

Nhập vào xâu kí tự **S**. Hãy đếm xem trong xâu **S** vừa nhập có bao nhiêu kí tự **9**

1.2. Ý tưởng giải: Duyệt từ đầu đến cuối xâu S, khi nào gặp kí tự **9** thì tăng biến đếm (**dem**) lên **1** đơn vị

1.3. Mô tả thuật toán:

- **Bước 1.** Nhập vào xâu kí tự **S**
- **Bước 2.** Khởi tạo biến **dem = 0**
- **Bước 3.** Duyệt từ đầu đến cuối xâu **S**, khi nào gặp kí tự **9** thì tăng biến **dem** lên **1** đơn vị **(dem = dem + 1)**
- **Bước 4.** In kết quả là biến **dem**

1.4. Code tham khảo (Python và C++):

1.4.1. Code Python:

```
fi = open('DEMKT.INP', 'r')          # Mở file để đọc
S = fi.read()                        # Đọc toàn bộ nội dung file vào chuỗi S
```

```
fi.close()                          # Đóng file sau khi đọc xong
dem = 0                             # Biến đếm số ký tự '9'
for i in range(0, len(S)):          # Duyệt từng ký tự trong chuỗi
    if S[i] == '9': dem = dem + 1   # Nếu ký tự là '9' thì tăng biến đếm lên
1
fo = open('DEMKT.OUT', 'w')         # Mở file "DEMK.OUT" để ghi kết quả
fo.write(f'{dem}')                  # Ghi kết quả (số lượng ký tự '9') vào
file
fo.close()                          # Đóng file sau khi ghi xong
```

1.4.2. Code C++:

```
#include <iostream>
#include <bits/stdc++.h>          // Thu vien tong hop
#include <fstream>                // Khai bao cai nay de su dung ifstream va
ofstream
using namespace std;
int main()
{
    ifstream fi("DEMKT.INP");     // Mo file de doc
    string S;
    getline(fi, S);               // Doc toan bo noi dung file vao chuoi S
    fi.close();                   // Dong file sau khi doc xong
    int dem = 0;                  // Bien dem so ky tu '9'
    for (int i = 0; i < S.length(); i++)     // Duyet tung ky tu trong
chuoi
    {
        if (S[i] == '9') dem = dem + 1;      // Neu ky tu la '9' thi tang
bien dem len 1
    }
    ofstream fo("DEMKT.OUT");     // Mo file "DEMK.OUT" de ghi ket qua
    fo << dem;                    // Ghi ket qua (so luong ky tu '9') vao file
    fo.close();                   // Dong file sau khi ghi xong
    return 0;
}
```

2. Bài 2. Siêu đối xứng *(2.0 điểm)*

Bài 2. Siêu đối xứng *(2.0 điểm)*

Một số nguyên dương được gọi là siêu đối xứng nếu tất cả các chữ số của nó giống nhau. Chẳng hạn số 777 hoặc 4444 là các số nguyên dương siêu đối xứng.

Nhập từ bàn phím một số nguyên dương x. Hãy tìm và in ra màn hình **số nguyên dương y nhỏ nhất** sao cho tổng $x + y$ là một số nguyên dương siêu đối xứng.

Ví dụ:

Nhập từ bàn phím	In ra màn hình
45	10

Giải thích: $45 + 10 = 55$

Giới hạn: Có 50% test ứng với $x \le 10^6$; 30% test ứng với $10^6 < x \le 10^9$; 20% test ứng với $10^9 \le x \le 10^{16}$

2.1. Tóm tắt đề bài:

Cho số nguyên dương **x**, tìm số nguyên dương nhỏ nhất **y** sao cho **x + y** là một số siêu đối xứng (nghĩa là tất cả các chữ số giống nhau, ví dụ: 1, 22, 333, 4444,...).

2.2. Ý tưởng giải:

- **Phân tích:**

+ Một số siêu đối xứng có dạng: d, dd, ddd, dddd, … (Với **d** là một chữ số từ **1** đến **9**)

+ Số siêu đối xứng được biểu diễn như sau:

sosieudoixung = d * (10^k – 1) / 9 (Với **k** là số chữ số)

+ Ở đây các số **d** và **k** có thể được hiểu như sau:

$1 \le d \le 9$

$1 \le k \le 18$ (Vì trong đề bài này, số **x** người ta cho giá trị là $x \le 10^{16}$ tức là **x** chỉ chứa tối đa **17** chữ số, nên phạm vi tối đa của số siêu đối xứng gần nhất cần tìm sẽ là **18** chữ số tương ứng với phạm vi của **k**)

- **Các giải:**

+ Sử dụng **2** vòng lặp **for** lồng nhau:

▪ Vòng **for** thứ **1**: Duyệt **k** chạy từ **1** đến **18**

▪ Vòng **for** thứ **2**: Duyệt **d** chạy từ **1** đến **9**

+ Sử dụng công thức sau để tính số siêu đối xứng trong **2** vòng **for** này:

sosieudoixung = d * (10^k – 1) // 9

+ Nếu **sosieudoixung** > **x** thì **y = sosieudoixung – x**

+ Giữ lại **y nhỏ nhất** trong tất cả các trường hợp **y** tính được

+ Xuất kết quả

2.3. Mô tả thuật toán:

- **Bước 1.** Nhập **x**
- **Bước 2.** Khởi tạo giá trị **y** nhỏ nhất ban đầu là **0**: **min_y = 10^18**
- **Bước 3.** Duyệt **2** vòng lặp **for** lồng nhau (**k** chạy từ **1** đến **18**, **d** chạy từ **1** đến **9**)
- **Bước 4.** Sử dụng công thức sau để tính số siêu đối xứng:

sosieudoixung = d * (10^k – 1) // 9

- **Bước 5.** Nếu **sosieudoixung** > **x** thì **y = sosieudoixung – x**
- **Bước 6.** Giữ lại giá trị **y** nhỏ nhất (**min_y**) trong tất cả các trường hợp **y** tính được
- **Bước 7.** Xuất ra kết quả của đề bài yêu cầu **min_y**

2.4. Code tham khảo (Python và C++):

2.4.1. Code Python:

```
fi = open('DX.INP', 'r')      # Mo file de doc
x = fi.read()                 # Doc x tu file
x = int(x)
fi.close()                    # Dong file sau khi doc xong
miny = 10**18                 # Khoi tao gia tri y nho nhat ban dau miny de so
sanh tim min
for k in range(1, 19):        # Duyet so chu so k cua so sieu doi xung tu 1 den
18 (1 chu so → 18 chu so)
```

```
    for d in range(1, 10):  # Duyet chu so d (1..9) – chu so lap trong so sieu
doi xung (khong co 0)
        sosieudoixung = d * (10**k - 1) // 9     # Tao so 'so sieu doi
xung' co k chu so, tat ca bang d
        if sosieudoixung > x:        # Neu so sieu doi xung lon hon x thi ta
co the dung no
            y = sosieudoixung - x    # Tinh y la so can cong de dat so sieu
doi xung do
            if y < miny: miny = y    # Neu y nho hon miny hien tai thi cap
nhat miny
fo = open('DX.OUT', 'w')     # Mo file de ghi
fo.write(f'{miny}')          # Ghi ket qua ra file
fo.close()                   # Dong file sau khi ghi xong
```

2.4.2. Code C++:

```
#include <iostream>
#include <bits/stdc++.h>     // Thu vien tong hop
#include <fstream>           // Khai bao cai nay de su dung ifstream va ofstream
using namespace std;
// Tao ham tinh 10 mu k
long long luythua10muk(int k)
{
    long long luythua = 1;
    for (int i = 0; i < k; i++)
        luythua = luythua * 10;
    return luythua;
}
int main()
{
    ifstream fi("DX.INP");  // Mo file de doc
    long long x;
    fi >> x;                // Doc x tu file
    fi.close();             // Dong file sau khi doc xong
    long long miny = 1e18;  // Khoi tao gia tri y nho nhat ban dau miny de so
sanh tim min
    for (int k = 1; k < 19; k++)         // Duyet so chu so k cua so sieu
doi xung tu 1 den 18 (1 chu so -> 18 chu so)
    {
        for (int d = 1; d < 10; d++)     // Duyet chu so d (1..9) – chu so
lap trong so sieu doi xung (khong co 0)
        {
            long long sosieudoixung = d * (luythua10muk(k) - 1) /
9;    // Tao so 'so sieu doi xung' co k chu so, tat ca bang d
            if (sosieudoixung > x)                    // Neu so sieu doi
xung lon hon x thi ta co the dung no
            {
                long long y = sosieudoixung - x;     // Tinh y la so can
cong de dat so sieu doi xung do
```

```
                if (y < miny) miny = y;                // Neu y nho hon
miny hien tai thi cap nhat miny
            }
        }
    }
    ofstream fo("DX.OUT");        // Mo file de ghi
    fo << miny;                   // Ghi ket qua ra file
    fo.close();                   // Dong file sau khi ghi xong
    return 0;
}
```

3. Bài 3. Giả thuyết Goldbach *(3.0 điểm)*

Bài 3. Giả thuyết Goldbach *(3.0 điểm)*

Giả thuyết Goldbach do nhà toán học người Đức Christian Goldbach (1690-1764) nêu ra vào năm 1742 trong một lá thư gửi tới nhà toán học Leonhard Euler, là một trong những bài toán lâu đời và nổi tiếng còn chưa giải được trong lý thuyết số nói riêng và toán học nói chung. Giả thuyết phỏng đoán rằng:

Mỗi số tự nhiên chẵn lớn hơn 2 có thể biểu diễn bằng tổng của hai số nguyên tố.

Giả thuyết đã được chỉ ra là đúng tới 4×10^{18}, nhưng hiện nay vẫn chưa được chứng minh hoàn toàn.

Bằng kiến thức tin học em hãy thực hiện thử thách sau:

Cho trước số x là số tự nhiên chẵn lớn hơn 2.

Yêu cầu: Hãy cho biết có bao nhiêu cách phân tích số x thành tổng của 2 số nguyên tố.

Lưu ý: Nếu phân tích được $x = a + b$ hay $x = b + a$ (trong đó a và b là các số nguyên tố) thì cũng chỉ được tính là một cách phân tích.

Input: Đọc ở file văn bản **GOL.INP** một số x là số tự nhiên chẵn lớn hơn 2.

Output: Ghi ra file văn bản **GOL.OUT** một số m theo yêu cầu.

Ví dụ:

GOL.INP	GOL.OUT
10	2

Giải thích: $10 = 3 + 7 = 5 + 5$

Giới hạn: Có 50% test ứng với $x \le 10^3$; 30% test ứng với $10^3 < x \le 10^4$; 20% test ứng với $10^4 < x \le 10^6$

3.1. Tóm tắt đề bài:

Đề bài yêu cầu: Đếm số cách phân tích một **số chẵn x > 2** thành tổng của **hai số nguyên tố** (Không phân biệt thứ tự, ví dụ 3 + 7 và 7 + 3 chỉ tính là 1 cách)

3.2. Ý tưởng giải:

- Dùng **hàm sàng nguyên tố** để đánh dấu các số nguyên tố đến **x**
- Duyệt **a** từ **2** đến **x // 2**, nếu cả **a** và **x – a** đều là số nguyên tố thì tăng biến đếm lên **1**
- Ghi kết quả đếm được ra file

3.3. Mô tả thuật toán:

- **Bước 1.** Tạo hàm sàng nguyên tố (**sangnguyento**)
- **Bước 2.** Nhập số chẵn **x > 2.**
- **Bước 3.** Dùng **sàng nguyên tố Eratosthenes** để tìm tất cả số nguyên tố **≤ x.**
- **Bước 4.** Khởi tạo biến **dem = 0.**
- **Bước 5.** Với **a** chạy từ **2** đến x **// 2:**

  Nếu **a** và **(x - a)** đều là số nguyên tố thì tăng biến **dem** lên **1** đơn vị **dem = dem + 1.**
- **Bước 6.** Ghi ra file giá trị **dem**.

3.4. Code tham khảo (Python và C++):

3.4.1. Code Python:

```
# Tạo hàm sàng nguyên tố
def sangnguyento(n):
    nguyento = [True] * (n + 1)          # Giả sử ban đầu mọi số đều là số
nguyên tố (True)
    nguyento[0] = nguyento[1] = False    # 0 và 1 không phải số nguyên tố ->
Gán False
    p = 2    # Bắt đầu với số nguyên tố nhỏ nhất là p = 2
    # Đánh dấu các bội của p để loại ra các số không phải là số nguyên tố
    # (Vì bội của p thì chia hết cho p, nên rõ ràng bội của p đúng là không phải số
nguyên tố)
    # Lặp p đến căn bậc hai của n, vì mọi bội nhỏ hơn p^2 đã bị đánh dấu loại trước đó
nên không cần xét nữa
    while p * p <= n:
        # Nếu p là số nguyên tố thì đánh dấu mọi bội của p (từ p^2 đến n) đều không
phải là số nguyên tố
        if nguyento[p]:
            for i in range(p * p, n + 1, p): nguyento[i] = False
        # Tiếp tục tăng p lên 1 đơn vị để kiểm tra và đánh dấu loại tiếp các số không
phải là số nguyên tố
        p = p + 1
    return nguyento # Trả về danh sách True/False tương ứng từng số
fi = open('GOL.INP', 'r')    # Mở file để đọc
x = fi.read()                # Đọc x từ file
x = int(x)
nguyento = sangnguyento(x)       # Tạo danh sách các số nguyên tố từ 0 đến x
dem = 0                          # Khởi tạo biến đếm số cách phân tích ban đầu
dem = 0
# Duyệt a từ 2 đến x//2 (chỉ cần nửa đầu vì a + b = x và (a,b) trùng với (b,a))
for a in range(2, x // 2 + 1):
    # Nếu a và (x - a) đều là số nguyên tố thì tăng biến đếm lên 1 đơn vị
```

```
    if nguyento[a] and nguyento[x - a]: dem = dem + 1
fo = open('GOL.OUT', 'w')    # Mở file để ghi
fo.write(f'{dem}')           # Ghi kết quả ra file
fo.close()                   # Đóng file sau khi ghi
```

3.4.2. Code C++:

```
#include <iostream>
#include <bits/stdc++.h>     // Thu vien tong hop
#include <fstream>           // Khai bao cai nay de su dung ifstream va ofstream
#include <vector>            // Khai bao cai nay de dung vector
using namespace std;
// Tao ham sang nguyen to
vector <bool> sangnguyento(int n)
{
    vector <bool> nguyento(n + 1, true);     // Gia su ban dau moi so deu
la so nguyen to (True)
    nguyento[0] = nguyento[1] = false;       // 0 va 1 khong phai la so
nguyen to -> Gan False
    int p = 2;                               // Bat dau voi so nguyen to
nho nhat la p = 2
    // Danh dau cac boi cua p de loai ra cac so khong phai la so nguyen to
    // (Vi boi cua p thi chia het cho p, nen ro rang boi cua p dung la khong phai so
nguyen to)
    // Lap p den can bac hai cua n, vi moi boi nho hon p^2 da bi danh dau loai truoc
do nen khong can xet nua
    while (p * p <= n)
    {
        // Neu p la so nguyen to thi danh dau moi boi cua p (tu p^2 den n) deu khong
phai la so nguyen to
        if (nguyento[p])
        {
            for (int i = p * p; i <= n; i += p)
                nguyento[i] = false;
        }
        // Tiep tuc tang p len 1 don vi de kiem tra va danh dau loai tiep cac so
khong phai la so nguyen to
        p = p + 1;
    }
    return nguyento;     // Tra ve danh sach True/False tuong ung tung so
}
int main()
{
    ifstream fi("GOL.INP"); // Mo file de doc
    int x;
    fi >> x;                 // Doc x tu file
    fi.close();              // Dong file sau khi doc
    vector <bool> nguyento = sangnguyento(x);     // Tao danh sach cac so
nguyen to tu 0 den x
```

```
    int dem = 0;                                        // Khoi tao bien dem so
cach phan tich ban dau dem = 0
    // Duyet a tu 2 den x/2 (Chi can nua dau vi a + b = x va (a,b) trung voi (b,a))
    for (int a = 2; a <= x / 2; a++)
    {
        // Neu a va (x - a) deu la so nguyen to thi tang bien dem len 1 don vi
        if (nguyento[a] && nguyento[x - a]) dem = dem +
1;
    }
    ofstream fo("GOL.OUT"); // Mo file de ghi
    fo << dem;              // Ghi ket qua ra file
    fo.close();             // Dong file sau khi ghi
    return 0;
}
```

4. Bài 4. Đường đi của Robot *(2.0 điểm)*

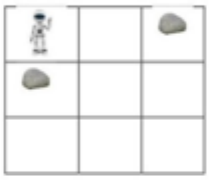
Bài 4. Đường đi của Robot *(2.0 điểm)*

Có một lưới ô vuông có kích thước $N \times N$ được đánh chỉ số hàng từ 1 đến N (theo chiều từ trên xuống dưới) và chỉ số cột từ 1 đến N (theo chiều từ trái sang phải). Mỗi ô trong lưới được xác định vị trí bởi một cặp số $(i; j)$ trong đó i là chỉ số hàng và j là chỉ số cột.

Tại ô $(1; 1)$ người ta đặt một con robot tự hành. Mỗi lần di chuyển robot chỉ đi sang phải một ô hoặc đi xuống dưới một ô. Trong lưới ô vuông này người ta đặt một viên đá vào một số ô để làm vật cản.

Yêu cầu: Hãy tính xem có bao nhiêu đường đi từ ô $(1; 1)$ đến ô $(N; N)$. Biết rằng robot không thể đi vào ô có vật cản và hai đường đi được gọi là khác nhau nếu có ít nhất một ô thuộc đường đi này nhưng không thuộc đường đi kia.

Vd: Xét lưới ô vuông kích thước 3x3 như hình vẽ sau:



Trong lưới ô vuông 3x3 này người ta đặt viên đá vào ô (1;3) và ô (2;1).
Với dữ kiện trên thì robot có tất cả 2 đường đi như sau:
(1;1) → (1;2) → (2;2) → (2;3) → (3;3)
(1;1) → (1;2) → (2;2) → (3;2) → (3;3)

Input: Đọc từ file văn bản **ROBOT.INP** có cấu trúc như sau:

- Dòng đầu tiên chứa 2 số nguyên dương N và M (mỗi số cách nhau 1 dấu cách; M < N).
- M dòng tiếp theo, mỗi dòng ghi 2 số nguyên dương i và j (mỗi số cách nhau một dấu cách) là chỉ số hàng và chỉ số cột của ô được đặt vào đó một viên đá là vật cản.

Output: Ghi ra file văn bản **ROBOT.OUT** một số k là số đường đi của robot từ ô $(1; 1)$ đến ô $(N; N)$.

Ví dụ:

ROBOT.INP	ROBOT.OUT
3 2 1 3 2 1	2

Giới hạn: 30% test ứng với $N \leq 10$; 40% test ứng với $10 < N \leq 30$; 30% test ứng với $30 < N \leq 100$;

4.1. Tóm tắt đề bài:

- Có một **lưới vuông kích thước N × N**.
- Một số ô trong lưới có **vật cản** (robot không thể đi qua).
- Robot bắt đầu từ ô **(1, 1)** (góc trên, bên trái) và cần đi đến ô **(N, N)** (góc dưới, bên phải).
- Robot **chỉ được đi xuống hoặc sang phải**.
- Cần tính xem **có bao nhiêu đường đi khác nhau** từ (1,1) đến (N,N) mà **không đụng vật cản**.

4.2. Ý tưởng giải: Dùng **quy hoạch động (Dynamic Programming – DP)** để **đếm số đường đi** đến mỗi ô (i, j)

- Gọi dp[i][j] là **số cách đi đến ô (i, j)**.
- Khi đó:
  + Nếu ô (i, j) là vật cản → dp[i][j] = 0
  + Nếu không: **dp[i][j] = dp[i − 1][j] + dp[i][j − 1]** (Robot có thể đến (i,j) từ trên xuống hoặc từ trái qua)
- Điều kiện biên:
  + dp[1][1] = 1 (nếu ô này không bị chặn)
  + Các ô ở hàng 1 hoặc cột 1 chỉ có thể đi theo 1 hướng.

4.3. Mô tả thuật toán:

- Bước 1: Nhập dữ liệu

+ Đọc `N` (kích thước lưới) và `M` (số lượng vật cản).
+ Tạo một **ma trận** `a` **kích thước (N+1)×(N+1)**, trong đó:
  - `a[i][j] = 0` nếu ô `(i, j)` trống.
  - `a[i][j] = 1` nếu ô `(i, j)` có vật cản.
+ Đọc tọa độ `M` ô có vật cản và gán `a[i][j] = 1`.

- Bước 2: Kiểm tra trường hợp đặc biệt

+ Nếu ô **bắt đầu (1,1)** hoặc **ô đích (N,N)** có vật cản, thì **không thể đi được** → kết quả là `0`.

- Bước 3: Khởi tạo mảng `dp`

+ Tạo ma trận `dp[N+1][N+1]` ban đầu toàn là 0.
+ `dp[i][j]` lưu **số đường đi từ (1,1) đến (i,j)**.
+ Gán `dp[1][1] = 1` vì chỉ có **1 cách duy nhất** để đứng ở ô xuất phát.

- Bước 4: Tính số đường đi đến từng ô

+ Duyệt qua từng ô `(i, j)` trong lưới theo thứ tự hàng và cột.
+ Với mỗi ô `(i, j)`:
  - Nếu **ô đó có vật cản** (`a[i][j] == 1`), bỏ qua không tính (`continue`).
  - Nếu không có vật cản, tính số đường đi đến `(i, j)` dựa vào:
    - **ô phía trên** `(i-1, j)` (đi xuống - có nghĩa là: từ ô phía trên),
    - **ô bên trái** `(i, j-1)` (đi sang phải - có nghĩa là: từ ô bên trái).
  - Khi đó: `dp[i][j] = dp[i - 1][j] + dp[i][j - 1]`
    - Nghĩa là: Số cách đến ô hiện tại = Số cách đến từ ô phía trên + Số cách đến từ ô bên trái.

- Bước 5: Kết quả

+ Sau khi duyệt hết lưới, kết quả cần tìm nằm ở: dp[N][N] → chính là **số đường đi từ (1,1) đến (N,N)**.

4.4. Code tham khảo (Python và C++):

4.4.1. Code Python:

```
# Mở file để đọc dữ liệu đầu vào
fi = open('ROBOT.INP', 'r')
# Đọc 2 số N và M từ dòng đầu tiên, N là kích thước lưới vuông N x N, M là số lượng vật cản
N, M = map(int, fi.readline().split())
# Khởi tạo mảng a ban đầu kích thước (N+1)x(N+1) chỉ chứa toàn giá trị 0
# Giá trị 0 là ô trống, 1 là ô có vật cản
a = [[0] * (N + 1) for _ in range(N + 1)]
# Đọc M dòng tiếp theo, mỗi dòng là tọa độ (i, j) của vật cản
for t in range(0, M):
    i, j = map(int, fi.readline().split())
    a[i][j] = 1      # Đánh dấu ô này có vật cản
fi.close()           # Đóng file sau khi đọc xong
# Nếu ô bắt đầu hoặc ô kết thúc có vật cản thì không có đường đi
if a[1][1] == 1 or a[N][N] == 1:
    fo = open('ROBOT.OUT', 'w')
    fo.write('0')    # Ghi ra số 0 (Không có đường đi)
    fo.close()
else:
    # Khởi tạo mảng dp kích thước (N+1)x(N+1)
    # dp[i][j] là số đường đi từ (1,1) đến (i,j)
    dp = [[0] * (N + 1) for _ in range(N + 1)]
    # Điểm bắt đầu (1,1) có 1 cách duy nhất
    dp[1][1] = 1
    # Tính số đường đi đến từng ô trong lưới
    for i in range(1, N + 1):            # Duyệt từng hàng
        for j in range(1, N + 1):        # Duyệt từng cột
            # Nếu ô này có vật cản thì bỏ qua, không tính toán
            if a[i][j] == 1:
                continue
            # Nếu không ở hàng đầu tiên thì cộng đường đi từ ô phía trên
            if i > 1:
                dp[i][j] = dp[i][j] + dp[i - 1][j]
            # Nếu không ở cột đầu tiên thì cộng đường đi từ ô bên trái
            if j > 1:
                dp[i][j] = dp[i][j] + dp[i][j - 1]
    fo = open('ROBOT.OUT', 'w') # Mở file để ghi kết quả
```

```
    fo.write(str(dp[N][N]))      # Ghi số đường đi từ (1,1) đến (N,N)
    fo.close()                   # Đóng file sau khi ghi xong
```

4.4.2. Code C++:

```
#include <iostream>
#include <bits/stdc++.h>     // Thu vien tong hop
#include <fstream>           // Khai bao cai nay de su dung ifstream va ofstream
using namespace std;
int main()
{
    // Mo file de doc du lieu vao
    ifstream fi("ROBOT.INP");
    // Doc 2 so N va M tu dong dau tien, N la kich thuoc luoi vuong N x N, M la so
luong vat can
    int N, M;
    fi >> N >> M;
    // Khoi tao mang a ban dau kich thuoc (N+1)x(N+1) chi chua toan gia tri 0
    // Gia tri 0 la o trong, 1 la o co vat can
    int a[101][101] = {0};
    // Doc M dong tiep theo, moi dong la toa do (i, j) cua vat can
    for (int t = 0; t < M; t++)
    {
        int i, j;
        fi >> i >> j;
        a[i][j] = 1;     // Danh dau o nay co vat can
    }
    fi.close();          // Dong file sau khi doc xong
    // Neu o bat dau hoac o ket thuc co vat can thi khong co duong di
    if (a[1][1] == 1 || a[N][N] == 1)
    {
        ofstream fo("ROBOT.OUT");
        fo << 0;         // Ghi ra so 0 (Khong co duong di)
        fo.close();
    }
    else
    {
        // Khoi tao mang dp kich thuoc (N+1)x(N+1)
        // dp[i][j] la so duong di tu (1,1) den (i,j)
        long long dp[101][101] = {0};
        // Diem bat dau (1,1) co 1 cach duy nhat
```

```
        dp[1][1] = 1;
        // Tinh so duong di den tung o trong luoi
        for (int i = 1; i <= N; i++)          // Duyet tung hang
        {
            for (int j = 1; j <= N; j++)      // Duyet tung cot
            {
                // Neu o nay co vat can thi bo qua, khong tinh toan
                if (a[i][j] == 1)
                    continue;
                // Neu khong o hang dau tien thi cong duong di tu o phia tren
                if (i > 1)
                    dp[i][j] = dp[i][j] + dp[i - 1][j];
                // Neu khong o cot dau tien thi cong duong di tu o ben trai
                if (j > 1)
                    dp[i][j] = dp[i][j] + dp[i][j - 1];
            }
        }
        ofstream fo("ROBOT.OUT");    // Mo file de ghi ket qua
        fo << dp[N][N];              // Ghi so duong di tu (1,1) den (N,N)
        fo.close();                  // Dong file sau khi ghi xong
    }
    return 0;
}
```

5. Các bộ Test:

Mỗi câu của đề thi đều có đủ 10 bộ Test, với các giá trị đầu vào tương ứng với tỉ lệ (%) của các ràng buộc (Substack) mà đề bài yêu cầu. Những bộ Test này đã được gửi ở tệp tin đính kèm cùng với tài liệu này.

6. Kết quả chấm thử bằng phần mềm Themis 1.9, giới hạn thời gian chạy của mỗi bộ Test là 01 giây:

Tên bài	Điểm tối đa	Điểm chấm thử bằng Themis 1.9	
		Python	C++
Bài 1. Đếm kí tự (DEMKT.*)	3.0 điểm	3.0 điểm (100%)	3.0 điểm (100%)
Bài 2. Siêu đối xứng (DX.*)	2.0 điểm	2.0 điểm (100%)	2.0 điểm (100%)
Bài 3. Giả thuyết Goldbach (GOL.*)	3.0 điểm	3.0 điểm (100%)	3.0 điểm (100%)
Bài 4. Đường đi của Robot (ROBOT.*)	2.0 điểm	2.0 điểm (100%)	1.4 điểm (70%)
Tổng số điểm đạt được:		**10.0 điểm (100%)**	**9.4 điểm (94%)**

ĐỀ SỐ 8
(THPT Chuyên Lê Quý Đôn – Đà Nẵng – Năm học: 2021 – 2022)

SỞ GIÁO DỤC VÀ ĐÀO TẠO
THÀNH PHỐ ĐÀ NẴNG

ĐỀ CHÍNH THỨC
(Đề thi có 02 trang)

KỲ THI TUYỂN SINH LỚP 10
TRUNG HỌC PHỔ THÔNG NĂM HỌC 2021-2022

Môn thi: **TIN HỌC** (chuyên)
Thời gian: **150** phút (không kể thời gian giao đề)
(Dành cho thí sinh thi vào Trường THPT chuyên Lê Quý Đôn)

TỔNG QUAN ĐỀ THI

Bài	Tên bài	File chương trình	Dữ liệu vào	Dữ liệu ra	Điểm
1	Tích lớn nhất	MMAX.*	MMAX.INP	MMAX.OUT	2,5
2	Biến đổi	CHANGE.*	CHANGE.INP	CHANGE.OUT	2,5
3	Số đặc biệt	SPRIME.*	SPRIME.INP	SPRIME.OUT	2,5
4	Số đối xứng	PALIND.*	PALIND.INP	PALIND.OUT	2,5

*(Phần mở rộng * là PAS hay CPP tùy theo ngôn ngữ và môi trường lập trình)*

Bài 1. (2,5 điểm) **Tích lớn nhất**

Cho dãy A gồm N số nguyên, hãy chọn ra 2 hoặc 3 phần tử sao cho tích của chúng là lớn nhất.

Dữ liệu vào: Từ file văn bản **MMAX.INP**

- Dòng đầu tiên: chứa số nguyên dương N ($3 \le N \le 10000$).
- Dòng thứ hai: chứa N số nguyên biểu diễn dãy A ($|Ai| \le 1000$).

Dữ liệu ra: Ghi ra file văn bản **MMAX.OUT**

Một số nguyên duy nhất là tích lớn nhất tìm được.

Ví dụ:

MMAX.INP	MMAX.OUT
6 5 10 -2 3 5 2	250

Bài 2. (2,5 điểm) **Biến đổi**

Cho dãy A gồm 8 số nguyên có giá trị từ 1 đến 8. Có 2 phép biến đổi trên dãy số này: Phép quay trái L và phép quay phải R.

Phép biến đổi L là đời số trong dãy từ phải sang trái, số đầu dãy chuyển đến vị trí cuối dãy.

Ví dụ: Dãy A: 12345678 Trạng thái dãy sau khi biến đổi L → 23456781

Tương tự, phép biến đổi R đời số trong dãy từ trái sang phải, số cuối dãy chuyển đến vị trí đầu dãy.

Ví dụ: Dãy A: 12345678 Trạng thái dãy sau khi biến đổi R → 81234567

Yêu cầu: Cho một dãy các phép biến đổi, sau khi thực hiện tuần tự các biến đổi đã cho, dãy A có trạng thái mới, biến đổi thành dãy B. Hãy lập trình xác định dãy B.

Dữ liệu vào: Từ file văn bản **CHANGE.INP**, chỉ gồm 1 hàng gồm các kí tự L, R viết liền nhau, dùng để biểu diễn dãy tuần tự các phép biến đổi cho trước. Chiều dài không quá 200 kí tự.

Dữ liệu ra: Ghi ra file văn bản **CHANGE.OUT**, gồm 1 dòng biểu diễn dãy B với các số viết liền nhau.

Ví dụ:

CHANGE.INP	CHANGE.OUT
RRRRRRR	23456781

Bài 3. (2,5 điểm) **Số đặc biệt**

Hiếu rất yêu thích số nguyên tố, đồng thời cũng rất yêu thích số 5. Hiếu luôn coi các số nguyên tố có tổng các chữ số chia hết cho 5 là số đặc biệt. Lần này, thầy giáo đưa cho Hiếu 2 số nguyên dương L, R ($L \le R$). Hiếu muốn biết trong đoạn [L,R] có bao nhiêu số đặc biệt nên nhờ các bạn trả lời giúp.

Dữ liệu vào: Từ file văn bản **SPRIME.INP**

- Dòng đầu tiên chứa số nguyên dương T ($1 \le T \le 100$) là số lượng test trong file.
- T dòng tiếp theo, mỗi dòng chứa hai số nguyên dương L, R ($0 < L \le R \le 3.10^6$) theo thứ tự, phân tách nhau bởi dấu cách.

Dữ liệu ra: Ghi ra file văn bản **SPRIME.OUT** T dòng, mỗi dòng ghi một số là số lượng số đặc biệt trong đoạn [L,R], tương ứng theo thứ tự trong file input. Dòng thứ i trong file output là kết quả của cặp số [L,R] ở dòng i+1 trong file input.

Ví dụ:

SPRIME.INP	SPRIME.OUT
2 1 10 4 20	1 2

Giải thích:

- *Trong đoạn [1, 10] có 1 số đặc biệt là 5.*
- *Trong đoạn [4, 20] có 2 số đặc biệt là 5 và 19 (1+9 = 10).*

Bài 4. (2,5 điểm) **Số đối xứng**

Số nguyên không âm được gọi là số palindrome nếu trong biểu diễn hệ 10 (không có số 0 không có nghĩa) đọc từ trái sang phải cho kết quả giống như đọc từ phải sang trái.

Ví dụ: Các số sau là palindrome: 5, 343, 66811866.

Yêu cầu: Cho số nguyên không âm X (X có không quá 10000 chữ số). Hãy tìm một số palindrome lớn nhất có trong X.

Dữ liệu vào: Từ file văn bản **PALIND.INP** gồm duy nhất một số nguyên X.

Dữ liệu ra: Ghi ra file văn bản **PALIND.OUT** gồm 2 dòng.

- Dòng đầu tiên ghi số chữ số của số palindrome lớn nhất có trong X.
- Dòng thứ 2 ghi số palindrome lớn nhất có trong X.

Ví dụ:

PALIND.INP	PALIND.OUT
21314	3 131
2131451516	3 515

----- Hết -----

BÀI GIẢI ĐỀ SỐ 8
(THPT Chuyên Lê Quý Đôn – Đà Nẵng – Năm học: 2021 – 2022)

1. Bài 1. (2,5 điểm) **Tích lớn nhất**

Bài 1. (2,5 điểm) **Tích lớn nhất**

Cho dãy A gồm N số nguyên, hãy chọn ra 2 hoặc 3 phần tử sao cho tích của chúng là lớn nhất.

Dữ liệu vào: Từ file văn bản **MMAX.INP**

- Dòng đầu tiên: chứa số nguyên dương N ($3 \le N \le 10000$).
- Dòng thứ hai: chứa N số nguyên biểu diễn dãy A ($|Ai| \le 1000$).

Dữ liệu ra: Ghi ra file văn bản **MMAX.OUT**

Một số nguyên duy nhất là tích lớn nhất tìm được.

Ví dụ:

MMAX.INP	MMAX.OUT
6 5 10 -2 3 5 2	250

1.1. Tóm tắt đề bài:

- Cho dãy số nguyên A gồm N phần tử ($3 \le N \le 10000$), mỗi phần tử có giá trị tuyệt đối ≤ 1000.
- Hãy chọn ra **2 hoặc 3 phần tử** trong dãy sao cho **tích của chúng là lớn nhất**.
- Kết quả là **một số nguyên duy nhất** - giá trị tích lớn nhất tìm được.
- Ví dụ:

MMAX.INP	MMAX.OUT	Giải thích
6 5 10 -2 3 5 2	250	Vì tích lớn nhất là 10 x 5 x 5 = 250

1.2. Ý tưởng giải:

- Tích lớn nhất có thể đến từ **4** trường hợp sau:
  + **Hai số dương lớn nhất**, hoặc
  + **Hai số âm nhỏ nhất (có trị tuyệt đối lớn)** → khi nhân với nhau cho kết quả dương, hoặc
  + **Ba số dương lớn nhất**, hoặc
  + **Hai số âm nhỏ nhất × một số dương lớn nhất** → cũng có thể tạo ra tích lớn.
- Vì vậy, ta chỉ cần **xét các phần tử lớn nhất và nhỏ nhất** trong dãy.

1.3. Mô tả thuật toán:

- **Bước 1.** Đọc số **N** và mảng **A**.
- **Bước 2. Sắp xếp mảng A tăng dần.**
- **Bước 3.** Gọi:
  + **a1, a2** là hai phần tử nhỏ nhất
  + **b1, b2, b3** là ba phần tử lớn nhất
- **Bước 4.** Tính bốn khả năng sau:
  + **kn1 = b3 * b2** → 2 số lớn nhất

+ **kn2 = a1 * a2** → 2 số âm nhỏ nhất
+ **kn3 = b3 * b2 * b1** → 3 số lớn nhất
+ **kn4 = a1 * a2 * b3** → 2 số âm nhỏ nhất nhân 1 số lớn nhất

- **Bước 5. Lấy giá trị lớn nhất trong các tích trên.**
- **Bước 6.** Ghi kết quả ra file **MMAX.OUT**.

1.4. Code tham khảo (Python và C++):

1.4.1. Code Python:

```
# Đọc dữ liệu từ file MMAX.INP
fi = open('MMAX.INP', 'r')                    # Mở file để đọc
N = int(fi.readline())                        # Đọc N
A = list(map(int, fi.readline().split()))     # Đọc mảng A
fi.close()                                    # Đóng file sau khi đọc xong
# Sắp xếp mảng A tăng dần
A.sort()
# Tính các khả năng có thể xảy ra
kn1 = A[N - 1] * A[N - 2]                 # 2 số lớn nhất
kn2 = A[0] * A[1]                         # 2 số nhỏ nhất
kn3 = A[N - 1] * A[N - 2] * A[N - 3]      # 3 số lớn nhất
kn4 = A[0] * A[1] * A[N - 1]              # 2 số nhỏ nhất và 1 số lớn nhất
# Lấy giá trị lớn nhất trong 4 khả năng trên, giá trị này chính là kết quả cần tìm
ketqua = max(kn1, kn2, kn3, kn4)
# Ghi kết quả ra file MMAX.OUT
fo = open('MMAX.OUT', 'w')   # Mở file để ghi
fo.write(f'{ketqua}')        # Ghi kết quả ra file
fo.close()                   # Đóng file sau khi ghi xong
```

1.4.2. Code C++:

```
#include <iostream>
#include <bits/stdc++.h>     // Thu vien tong hop
#include <fstream>           // Khai bao cai nay de doc, ghi file
#include <vector>            // Khai bao cai nay de dung vector
using namespace std;
int main()
{
// Doc du lieu tu file MMAX.INP
ifstream fi("MMAX.INP");                  // Mo file de doc
int N;
fi >> N;                                  // Doc N
vector<int> A(N);
for (int i = 0; i < N; i++) fi >> A[i]; // Doc mang A
fi.close();                               // Dong file sau khi doc xong
// Sap xep mang A tang dan
sort(A.begin(), A.end());
// Tinh cac kha nang co the xay ra
long long kn1 = 1LL * A[N-1] * A[N-2];           // 2 so lon nhat
long long kn2 = 1LL * A[0] * A[1];               // 2 so nho nhat
long long kn3 = 1LL * A[N-1] * A[N-2] * A[N-3]; // 3 so lon nhat
```

```
long long kn4 = 1LL * A[0] * A[1] * A[N-1];      // 2 so nho nhat va 1 so
lon nhat
// Lay gia tri lon nhat trong 4 kha nang tren, gia tri nay chinh la ket qua can tim
long long ketqua = max({kn1, kn2, kn3, kn4});
// Ghi ket qua ra file MMAX.OUT
ofstream fo("MMAX.OUT");     // Mo file de ghi
fo << ketqua;                // Ghi ket qua ra file
fo.close();                  // Dong file sau khi ghi xong
return 0;
}
```

2. Bài 2. (2,5 điểm) **Biến đổi**

Bài 2. (2,5 điểm) **Biến đổi**

Cho dãy A gồm 8 số nguyên có giá trị từ 1 đến 8. Có 2 phép biến đổi trên dãy số này: Phép quay trái L và phép quay phải R.

Phép biến đổi L là dời số trong dãy từ phải sang trái, số đầu dãy chuyển đến vị trí cuối dãy.

Ví dụ: Dãy A: 12345678 Trạng thái dãy sau khi biến đổi L → 23456781

Tương tự, phép biến đổi R dời số trong dãy từ trái sang phải, số cuối dãy chuyển đến vị trí đầu dãy.

Ví dụ: Dãy A: 12345678 Trạng thái dãy sau khi biến đổi R → 81234567

Yêu cầu: Cho một dãy các phép biến đổi, sau khi thực hiện tuần tự các biến đổi đã cho, dãy A có trạng thái mới, biến đổi thành dãy B. Hãy lập trình xác định dãy B.

Dữ liệu vào: Từ file văn bản **CHANGE.INP**, chỉ gồm 1 hàng gồm các kí tự L, R viết liền nhau, dùng để biểu diễn dãy tuần tự các phép biến đổi cho trước. Chiều dài không quá 200 kí tự.

Dữ liệu ra: Ghi ra file văn bản **CHANGE.OUT**, gồm 1 dòng biểu diễn dãy B với các số viết liền nhau.

Ví dụ:

CHANGE.INP	CHANGE.OUT
RRRRRRR	23456781

2.1. Tóm tắt đề bài:

- Cho dãy **A** gồm **8 số nguyên** (từ 1 đến 8).
- Có 2 phép biến đổi:
  + **L**: quay trái → phần tử đầu tiên ra cuối dãy.
  + **R**: quay phải → phần tử cuối cùng ra đầu dãy.
- Cho một chuỗi **S** gồm các ký tự **L** và **R** (chiều dài ≤ 200).
- Thực hiện các phép biến đổi **tuần tự từ trái sang phải** trên dãy **A** ban đầu = 12345678.
- Xuất ra trạng thái cuối cùng của dãy sau khi thực hiện hết.
- Ví dụ:
  + Input: **RRRRRRR**
  + Output: **23456781**

2.2. Ý tưởng giải:

- Dãy **A** ban đầu luôn là "12345678".
- Mỗi ký tự trong chuỗi lệnh điều khiển một phép quay:
  + Nếu là **'L'**: Cắt ký tự **đầu** của dãy **A**, ghép vào **cuối** dãy **A**.
  + Nếu là **'R'**: Cắt ký tự **cuối** của dãy **A**, ghép vào **đầu** dãy **A**.
- Có thể thực hiện tuần tự từng lệnh trên chuỗi.

2.3. Mô tả thuật toán:

- **Bước 1.** Đọc dữ liệu từ file **CHANGE.INP** → 1 xâu **S** chứa chuỗi các ký tự **L, R**.
- **Bước 2. Khởi tạo dãy A = "12345678"**.
- **Bước 3. Duyệt từng ký tự** trong chuỗi **S**:
  + Nếu là **'L'**: Cắt ký tự **đầu** của dãy **A**, ghép vào **cuối** dãy **A**.
  + Nếu là **'R'**: Cắt ký tự **cuối** của dãy **A**, ghép vào **đầu** dãy **A**.
- **Bước 4.** Sau khi duyệt hết, ghi kết quả ra file **CHANGE.OUT**.

2.4. Code tham khảo (Python và C++):

2.4.1. Code Python:

```python
# Doc du lieu tu file CHANGE.INP
fi = open('CHANGE.INP', 'r')      # Mo file de doc
S = fi.readline().strip()         # Doc chuoi S cac phep bien doi (L hoac R)
fi.close()                        # Dong file sau khi doc xong
# Khoi tao day A ban dau
A = "12345678"
# Thuc hien cac phep bien doi
for kt in S:
    if kt == 'L':
        # Quay trai: So dau tien chuyen ra cuoi day
        dau = A[0]
        phan_con_lai = ""
        for i in range(1, len(A)):
            phan_con_lai = phan_con_lai + A[i]
        A = phan_con_lai + dau
    elif kt == 'R':
        # Quay phai: So cuoi cung chuyen ra dau day
        cuoi = A[len(A) - 1]
        phan_con_lai = ""
        for i in range(0, len(A) - 1):
            phan_con_lai = phan_con_lai + A[i]
        A = cuoi + phan_con_lai
# Ghi ket qua ra file CHANGE.OUT
fo = open('CHANGE.OUT', 'w')      # Mo file de ghi
fo.write(A)                       # Ghi ket qua
fo.close()                        # Dong file sau khi ghi xong
```

2.4.2. Code C++:

```cpp
#include <iostream>
#include <bits/stdc++.h>     // Thu vien tong hop
#include <fstream>           // Khai bao cai nay de doc, ghi file
```

```
using namespace std;
int main()
{
// Doc du lieu tu file CHANGE.INP
ifstream fi("CHANGE.INP");   // Mo file de doc
string S;
fi >> S;                     // Doc chuoi S cac phep bien doi (L hoac R)
fi.close();                  // Dong file sau khi doc xong
// Khoi tao day A ban dau
string A = "12345678";
// Thuc hien cac phep bien doi
for (int k = 0; k < (int)S.size(); k++)
{
    char kt = S[k];
    if (kt == 'L')
    {
        // Quay trai: So dau tien chuyen ra cuoi day
        char dau = A[0];
        string phan_con_lai = "";
        for (int i = 1; i < (int)A.size(); i++)
        {
            phan_con_lai = phan_con_lai + A[i];
        }
        A = phan_con_lai + dau;
    }
    else if (kt == 'R')
    {
        // Quay phai: So cuoi cung chuyen ra dau day
        char cuoi = A[A.size() - 1];
        string phan_con_lai = "";
        for (int i = 0; i < (int)A.size() - 1; i++)
        {
            phan_con_lai = phan_con_lai + A[i];
        }
        A = cuoi + phan_con_lai;
    }
}
// Ghi ket qua ra file CHANGE.OUT
ofstream fo("CHANGE.OUT");   // Mo file de ghi
fo << A;                     // Ghi ket qua
fo.close();                  // Dong file sau khi ghi xong
return 0;
}
```

3. Bài 3. (2,5 điểm) Số đặc biệt

Bài 3. (2,5 điểm) **Số đặc biệt**

Hiếu rất yêu thích số nguyên tố, đồng thời cũng rất yêu thích số 5. Hiếu luôn coi các số nguyên tố có tổng các chữ số chia hết cho 5 là số đặc biệt. Lần này, thầy giáo đưa cho Hiếu 2 số nguyên dương L, R (L ≤ R). Hiếu muốn biết trong đoạn [L,R] có bao nhiêu số đặc biệt nên nhờ các bạn trả lời giúp.

Dữ liệu vào: Từ file văn bản **SPRIME.INP**

- Dòng đầu tiên chứa số nguyên dương T ($1 \le T \le 100$) là số lượng test trong file.
- T dòng tiếp theo, mỗi dòng chứa hai số nguyên dương L, R ($0 < L \le R \le 3.10^6$) theo thứ tự, phân tách nhau bởi dấu cách.

Dữ liệu ra: Ghi ra file văn bản **SPRIME.OUT** T dòng, mỗi dòng ghi một số là số lượng số đặc biệt trong đoạn [L,R], tương ứng theo thứ tự trong file input. Dòng thứ i trong file output là kết quả của cặp số [L,R] ở dòng i+1 trong file input.

Ví dụ:

SPRIME.INP	SPRIME.OUT
2 1 10 4 20	1 2

Giải thích:

- Trong đoạn [1, 10] có 1 số đặc biệt là 5.

- Trong đoạn [4, 20] có 2 số đặc biệt là 5 và 19 (1+9 = 10).

3.1. Tóm tắt đề bài:

- Hiếu yêu thích **số 5** và **số nguyên tố**.
- Hiếu gọi một **số đặc biệt** là **số nguyên tố có tổng các chữ số chia hết cho 5**.
- Cho T bộ test, mỗi test gồm hai số nguyên dương L, R.
- **Yêu cầu:** Với mỗi cặp [L, R], hãy đếm có bao nhiêu **số đặc biệt** trong đoạn này.
- **Giới hạn:**

  $\mathbf{1 \le T \le 100}$

  $\mathbf{0 < L \le R \le 3 \times 10^6}$

3.2. Ý tưởng giải:

- Ta cần kiểm tra trong đoạn [L, R]:
  + Số nào là **số nguyên tố**.
  + **Tổng các chữ số** của số nguyên tố đó **chia hết cho 5**.
- Để làm nhanh:
  + Dùng **sàng Eratosthenes** để đánh dấu các số nguyên tố $\le 3\times10^6$ một lần duy nhất.
  + Sau đó, duyệt qua các số trong đoạn [L, R], nếu là nguyên tố và tổng chữ số chia hết cho 5 thì tăng biến đếm lên 1 đơn vị.

3.3. Mô tả thuật toán:

- **Bước 1. Tạo hàm sàng nguyên tố** Eratosthenes.
- **Bước 2.** Tạo hàm tính tổng các chữ số của **n**.

- **Bước 3.** Đọc số lượng test **T**.
- **Bước 4.** Với mỗi test, đọc cặp **(L, R)**.
- **Bước 5. Dùng sàng Eratosthenes** để tạo mảng `nguyento[n]` (**True** nếu **n** là số nguyên tố).
- **Bước 6.** Với mỗi đoạn **[L, R]**:
  - Đặt `dem = 0`
  - Duyệt `i` từ `L` đến `R`:
    + Nếu `nguyento[i]` là **True**:
      ▪ Tính tổng các chữ số của `i`
      ▪ Nếu tổng này **chia hết cho 5** → `dem = dem + 1`
- **Bước 7.** Ghi ra `dem`.

3.4. Code tham khảo (Python và C++):

3.4.1. Code Python:

```
# Tạo hàm sàng nguyên tố Eratosthenes để đánh dấu số nguyên tố
def sangnguyento(n):
    nguyento = [True] * (n + 1)          # Giả sử ban đầu mọi số đều là số
nguyên tố (True)
    nguyento[0] = nguyento[1] = False    # 0 và 1 không phải số nguyên tố ->
Gán False
    p = 2    # Bắt đầu với số nguyên tố nhỏ nhất là p = 2
    # Đánh dấu các bội của p để loại ra các số không phải là số nguyên tố
    # (Vì bội của p thì chia hết cho p, nên rõ ràng bội của p đúng là không phải số
nguyên tố)
    # Lặp p đến căn bậc hai của n, vì mọi bội nhỏ hơn p^2 đã bị đánh dấu loại trước đó
nên không cần xét nữa
    while p * p <= n:
        # Nếu p là số nguyên tố thì đánh dấu mọi bội của p (từ p^2 đến n) đều không
phải là số nguyên tố
        if nguyento[p]:
            for i in range(p * p, n + 1, p): nguyento[i] = False
        # Tiếp tục tăng p lên 1 đơn vị để kiểm tra và đánh dấu loại tiếp các số không
phải là số nguyên tố
        p = p + 1
    return nguyento # Trả về danh sách True/False tương ứng từng số
# Tạo hàm tính tổng các chữ số
def tongchuso(x):
    tong = 0
    while x > 0:
        tong = tong + x % 10
        x = x // 10
    return tong
# Đọc dữ liệu từ file SPRIME.INP
fi = open('SPRIME.INP', 'r')             # Mở file để đọc
T_line = fi.readline().strip()           # Đọc dòng đầu tiên
while T_line == "":                      # Nếu dòng đầu tiên rỗng (dòng
trống), đọc lại
```

```
    T_line = fi.readline().strip()
T = int(T_line)
# Nhập số lượng test T
LR = []      # Mảng lưu các cặp test (L, R)
Rmax = 0     # Biến lưu R lớn nhất
for i in range(0, T):
    line = fi.readline().strip()          # Đọc từng dòng, bỏ qua khoảng
trắng
    while line == "":                     # Nếu dòng trống, đọc tiếp (tránh
lỗi dư dòng)
        line = fi.readline().strip()
    L, R = map(int, line.split())         # Đọc 2 số L, R
    LR.append((L, R))                     # Thêm cặp (L, R) vào danh sách
    if R > Rmax: Rmax = R                 # Cập nhật lại Rmax
fi.close()                                # Đóng file sau khi đọc xong
# Dùng sàng Eratosthenes để tạo mảng nguyento[] từ 0 đến giá trị lớn nhất Rmax
nguyento = sangnguyento(Rmax)
# Ghi kết quả ra file SPRIME.OUT
fo = open('SPRIME.OUT', 'w')     # Mở file để ghi kết quả
for L, R in LR:                  # Duyệt từng cặp L, R
    dem = 0                      # Khởi tạo biến dem = 0
    for i in range(L, R + 1):    # Duyệt từ L đến R
        if nguyento[i] and tongchuso(i) % 5 == 0:    # Nếu thỏa điều kiện
đề bài
            dem = dem + 1    # Tăng biến đếm lên 1 đơn
vị
    fo.write(f"{dem}\n")     # Ghi kết quả ra file
fo.close()  # Đóng file sau khi ghi
xong
```

3.4.2. Code C++:

```
#include <iostream>
#include <bits/stdc++.h>     // Thu vien tong hop
#include <fstream>           // Khai bao cai nay de su dung ifstream va ofstream
#include <vector>            // Khai bao cai nay de dung vector
using namespace std;
// Tao ham sang nguyen to Eratosthenes de danh dau so nguyen to
vector <bool> sangnguyento(int n)
{
    vector <bool> nguyento(n + 1, true);     // Gia su ban dau moi so deu
la so nguyen to (True)
    nguyento[0] = nguyento[1] = false;       // 0 va 1 khong phai la so
nguyen to -> Gan False
    int p = 2;                               // Bat dau voi so nguyen to
nho nhat la p = 2
    // Danh dau cac boi cua p de loai ra cac so khong phai la so nguyen to
```

```
    // (Vi boi cua p thi chia het cho p, nen ro rang boi cua p dung la khong phai so
nguyen to)
    // Lap p den can bac hai cua n, vi moi boi nho hon p^2 da bi danh dau loai truoc
do nen khong can xet nua
    while (p * p <= n)
    {
        // Neu p la so nguyen to thi danh dau moi boi cua p (tu p^2 den n) deu khong
phai la so nguyen to
        if (nguyento[p])
        {
            for (int i = p * p; i <= n; i += p)
                nguyento[i] = false;
        }
        // Tiep tuc tang p len 1 don vi de kiem tra va danh dau loai tiep cac so
khong phai la so nguyen to
        p = p + 1;
    }
    return nguyento;     // Tra ve danh sach True/False tuong ung tung so
}
// Tao ham tinh tong cac chu so
int tongchuso(int x)
{
    int tong = 0;
    while (x > 0)
    {
        tong = tong + x % 10;
        x = x / 10;
    }
    return tong;
}
int main()
{
    // Doc du lieu tu file SPRIME.INP
    ifstream fi("SPRIME.INP");  // Mo file de doc
    string T_line;
    getline(fi, T_line);        // Doc dong dau tien
    while (T_line == "") getline(fi, T_line); // Neu dong dau tien rong
(dong trong), doc lai
    int T = stoi(T_line);       // Nhap so luong test T
    vector<pair<int,int>> LR;   // Mang luu cac cap test (L, R)
    int Rmax = 0;               // Bien luu R lon nhat
    for (int i = 0; i < T; i++)
    {
        string line;
        getline(fi, line); // Doc tung dong, bo qua khoang trang
        while (line == "") getline(fi, line); // Neu dong trong, doc tiep
(tranh loi du dong)
```

```
        stringstream ss(line);
        int L, R;
        ss >> L >> R;              // Doc 2 so L, R
        LR.push_back({L, R});      // Them cap (L, R) vao danh sach
        if (R > Rmax) Rmax = R; // Cap nhat Rmax
    }
    fi.close(); // Dong file sau khi doc xong
    // Dung sang Eratosthenes de tao mang nguyento[] tu 0 den gia tri lon nhat Rmax
    vector<bool> nguyento = sangnguyento(Rmax);
    // Ghi ket qua ra file SPRIME.OUT
    ofstream fo("SPRIME.OUT"); // Mo file de ghi ket qua
    for (auto cap : LR)
    {
        int L = cap.first;
        int R = cap.second;
        int dem = 0; // Khoi tao bien dem = 0
        for (int i = L; i <= R; i++)     // Duyet tu L den R
        {
            if (nguyento[i] && tongchuso(i) % 5 == 0) // Neu thoa dieu
kien de bai
                dem = dem + 1; // Tang bien dem len 1 don vi
        }
        fo << dem << "\n"; // Ghi ket qua ra file
    }
    fo.close(); // Dong file sau khi ghi xong
    return 0;
}
```

4. Bài 4. (2,5 điểm) Số đối xứng

Bài 4. (2,5 điểm) **Số đối xứng**

Số nguyên không âm được gọi là số palindrome nếu trong biểu diễn hệ 10 (không có số 0 không có nghĩa) đọc từ trái sang phải cho kết quả giống như đọc từ phải sang trái.

Ví dụ: Các số sau là palindrome: 5, 343, 66811866.

Yêu cầu: Cho số nguyên không âm X (X có không quá 10000 chữ số). Hãy tìm một số palindrome lớn nhất có trong X.

Dữ liệu vào: Từ file văn bản **PALIND.INP** gồm duy nhất một số nguyên X.

Dữ liệu ra: Ghi ra file văn bản **PALIND.OUT** gồm 2 dòng.

- Dòng đầu tiên ghi số chữ số của số palindrome lớn nhất có trong X.
- Dòng thứ 2 ghi số palindrome lớn nhất có trong X.

Ví dụ:

PALIND.INP	PALIND.OUT
21314	3 131
2131451516	3 515

4.1. Tóm tắt đề bài:

- Cho một số nguyên không âm **X** (không quá 10.000 chữ số).
- Tìm **số đối xứng (palindrome)** có **độ dài lớn nhất** nằm **bên trong X** (tức là một đoạn con của X).
- Xuất ra:
  + Dòng 1: Độ dài của số đối xứng dài nhất.
  + Dòng 2: Giá trị của số đối xứng lớn nhất có cùng độ dài đó.
- Ví dụ:
  + Input: **2131451516**
  + Output:
    3
    515

4.2. Ý tưởng giải:

- Một **số đối xứng** là chuỗi ký tự đọc xuôi và đọc ngược như nhau.
- Bài yêu cầu **tìm đoạn con đối xứng dài nhất** trong chuỗi ký tự biểu diễn của X.
- Sau đó, **nếu có nhiều đoạn con cùng độ dài**, chọn đoạn con có **giá trị số lớn nhất**.
- **Ý tưởng chính:**
  + Duyệt qua từng vị trí trong chuỗi (mỗi vị trí có thể là tâm của số đối xứng).
  + Mở rộng ra hai phía (trái và phải) để kiểm tra đối xứng.
  + Làm điều đó cho:
    - Palindrome có độ dài lẻ (1 ký tự làm tâm)
    - Palindrome có độ dài chẵn (2 ký tự làm tâm)
  + Cập nhật độ dài lớn nhất và giá trị tương ứng.

4.3. Mô tả thuật toán:

- **Bước 1. Tạo hàm mở rộng tâm.** Hàm mở rộng từ vị trí tâm ra hai bên để kiểm tra xem có phải số đối xứng không.
- **Bước 2.** Đọc chuỗi **X** từ file **PALIND.INP**.
- **Bước 3.** Khởi tạo:
  + dodaimax = 0
  + sodoixungmax = ""
- **Bước 4.** Với mỗi vị trí **i** trong chuỗi **X**:
  + **Trường hợp lẻ**: Mở rộng quanh **i** (ví dụ: **X[i-1], X[i+1]**)
  + **Trường hợp chẵn**: Mở rộng quanh **i** và **i+1**
- **Bước 5.** Trong mỗi lần mở rộng:
  + Khi hai ký tự ở hai đầu còn giống nhau, tiếp tục mở rộng.

+ Nếu phát hiện palindrome dài hơn **dodaimax**, cập nhật **dodaimax** và **sodoixungmax**.

+ Nếu độ dài bằng nhau, so sánh giá trị số **(int(sodoixungmax))** để lấy số lớn hơn.

- **Bước 5.** Sau khi duyệt hết, ghi kết quả ra file **PALIND.OUT**:

+ Dòng 1: **dodaimax**

+ Dòng 2: s**odoixungmax**

4.4. Code tham khảo (Python và C++):

4.4.1. Code Python:

```
# Tạo hàm mở rộng tâm từ vị trí tâm ra hai bên để kiểm tra xem có phải số đối xứng
không
def morongtam(chuoi, trai, phai):
    while trai >= 0 and phai < len(chuoi) and chuoi[trai] ==
chuoi[phai]:
        trai = trai - 1              # Di chuyển sang trái
        phai = phai + 1              # Di chuyển sang phải
    return chuoi[trai + 1: phai]     # Trả về đoạn đối xứng tìm được
# Đọc dữ liệu từ file PALIND.
fi = open('PALIND.INP', 'r')     # Mở file để đọc
X = fi.readline().strip()        # Đọc chuỗi số X và loại bỏ ký tự xuống dòng
fi.close()                       # Đóng file sau khi đọc xong
dodaimax = 0                     # Khởi tạo biến lưu độ dài lớn nhất của số đối
xứng
sodoixungmax = ""                # Khởi tạo biến lưu chuỗi số đối xứng lớn nhất
cần tìm
# Duyệt qua từng vị trí trong chuỗi để làm tâm mở rộng
for i in range(0, len(X)):
    # Trường hợp đối xứng lẻ (một ký tự làm tâm)
    doixungle = morongtam(X, i, i)
    # Trường hợp đối xứng chẵn (hai ký tự liền nhau làm tâm)
    doixungchan = morongtam(X, i, i + 1)
    # Kiểm tra và cập nhật kết quả với số đối xứng lẻ
    if len(doixungle) > dodaimax or (len(doixungle) == dodaimax and
doixungle > sodoixungmax):
        dodaimax = len(doixungle)
        sodoixungmax = doixungle
    # Kiểm tra và cập nhật kết quả với số đối xứng chẵn
    if len(doixungchan) > dodaimax or (len(doixungchan) == dodaimax
and doixungchan > sodoixungmax):
        dodaimax = len(doixungchan)
        sodoixungmax = doixungchan
# Ghi kết quả ra file PALIND.OUT
```

```
fo = open('PALIND.OUT', 'w')      # Mở file để ghi
fo.write(f'{dodaimax}\n')         # Dòng 1: độ dài của số đối xứng lớn nhất
fo.write(sodoixungmax)            # Dòng 2: nội dung số đối xứng lớn nhất
fo.close()                        # Đóng file sau khi ghi xong
```

4.4.2. Code C++:

```
#include <iostream>
#include <bits/stdc++.h>     // Thu vien tong hop
#include <fstream>           // Khai bao cai nay de su dung ifstream va ofstream
#include <string>            // Khai bao cai nay de xu li chuoi
using namespace std;
// Tao ham mo rong tam tu vi tri tam ra hai ben de kiem tra xem co phai so doi xung
khong
string morongtam(string chuoi, int trai, int phai)
{
    while (trai >= 0 && phai < (int)chuoi.size() && chuoi[trai] ==
chuoi[phai])
    {
        trai = trai - 1;     // Di chuyen sang trai
        phai = phai + 1;     // Di chuyen sang phai
    }
    return chuoi.substr(trai + 1, phai - trai - 1); // Tra ve doan doi
xung tim duoc
}
int main()
{
    // Doc du lieu tu file PALIND.INP
    ifstream fi("PALIND.INP");  // Mo file de doc
    string X;
    getline(fi, X);             // Doc chuoi so X va loai bo ky tu xuong dong
    fi.close();                 // Dong file sau khi doc xong
    int dodaimax = 0;           // Khoi tao bien luu do dai lon nhat cua so
doi xung
    string sodoixungmax = "";   // Khoi tao bien luu chuoi so doi xung lon
nhat tim duoc
    // Duyet qua tung vi tri trong chuoi de lam tam mo rong
    for (int i = 0; i < (int)X.size(); i++)
    {
        // Truong hop doi xung le (mot ky tu lam tam)
        string doixungle = morongtam(X, i, i);
        // Truong hop doi xung chan (hai ky tu lien nhau lam tam)
        string doixungchan = morongtam(X, i, i + 1);
```

```
        // Kiem tra va cap nhat ket qua voi so doi xung le
        if ((int)doixungle.size() > dodaimax ||
((int)doixungle.size() == dodaimax && doixungle > sodoixungmax))
        {
            dodaimax = doixungle.size();
            sodoixungmax = doixungle;
        }
        // Kiem tra va cap nhat ket qua voi so doi xung chan
        if ((int)doixungchan.size() > dodaimax ||
((int)doixungchan.size() == dodaimax && doixungchan >
sodoixungmax))
        {
            dodaimax = doixungchan.size();
            sodoixungmax = doixungchan;
        }
    }
    // Ghi ket qua ra file PALIND.OUT
    ofstream fo("PALIND.OUT");        // Mo file de ghi
    fo << dodaimax << "\n";           // Dong 1: do dai cua so doi xung lon
nhat
    fo << sodoixungmax;               // Dong 2: in ra chuoi so doi xung lon
nhat
    fo.close();                       // Dong file sau khi ghi xong
    return 0;
}
```

5. Các bộ Test:

Mỗi câu của đề thi đều có đủ 10 bộ Test, với các giá trị đầu vào tương ứng với tỉ lệ (%) của các ràng buộc (Substack) mà đề bài yêu cầu. Những bộ Test này đã được gửi ở tệp tin đính kèm cùng với tài liệu này.

6. Kết quả chấm thử bằng phần mềm Themis 1.9, giới hạn thời gian chạy của mỗi bộ Test là 01 giây:

Tên bài	Điểm tối đa	Điểm chấm bằng Themis 1.9	
		Python	C++
Bài 1. Tích lớn nhất (MMAX.*)	2.5 điểm	2.5 điểm (100%)	2.5 điểm (100%)
Bài 2. Biến đổi (CHANGE.*)	2.5 điểm	2.5 điểm (100%)	2.5 điểm (100%)
Bài 3. Số đặc biệt (SPRIME.*)	2.5 điểm	1.0 điểm (40%)	2.5 điểm (100%)
Bài 4. Số đối xứng (PALIND.*)	2.5 điểm	2.5 điểm (100%)	2.5 điểm (100%)
Tổng số điểm đạt được:		**8.5 điểm (85%)**	**10.0 điểm (100%)**

ĐỀ SỐ 9
(Học sinh giỏi 9 – Đà Nẵng – Năm học: 2020 – 2021)

SỞ GIÁO DỤC VÀ ĐÀO TẠO
THÀNH PHỐ ĐÀ NẴNG

KỈ THI CHỌN HỌC SINH GIỎI LỚP 9 CẤP THÀNH PHỐ
NĂM HỌC 2020 - 2021

MÔN: TIN HỌC

ĐỀ CHÍNH THỨC

Thời gian làm bài: 150 phút *(không kể thời gian giao đề)*
(Đề thi có 3 trang)

TỔNG QUAN

	Tên bài	File chương trình	File dữ liệu vào	File dữ liệu ra
Câu 1	Số nguyên tố cân bằng	NGUYENTO.*	Nhập từ bàn phím	In ra màn hình
Câu 2	Từ đại diện	DAIDIEN.*	DAIDIEN.INP	DAIDIEN.OUT
Câu 3	Diện tích lớn nhất	DIENTICH.*	DIENTICH.INP	DIENTICH.OUT

*Chú ý: Dấu * có thể là pas; c; cpp hoặc py tương ứng với ngôn ngữ lập trình là Free Pacal; C/C++ hoặc Python.*

Câu 1. Số nguyên tố cân bằng (3 điểm)

Một số được gọi là số nguyên tố cân bằng nếu nó là số nguyên tố có 2k + 1 chữ số ($k \in N^*$), trong đó có 2k chữ số giống nhau và có đúng một chữ số ở vị trí chính giữa (tức vị trí thứ k + 1 tính từ trái sang phải) là khác với các chữ số còn lại. Chẳng hạn số 7778777 là số nguyên tố cân bằng.

Yêu cầu:

Nhập từ bàn phím một số nguyên dương k ($k \in N^*, k \le 7$). Hãy tính và in ra màn hình số lượng các số nguyên tố cân bằng có 2k + 1 chữ số.

Ví dụ:

Nhập từ bàn phím	In ra màn hình
3	7

Giải thích: Có 7 số nguyên tố cân bằng có 7 chữ số: 1114111; 1117111; 3331333; 3337333; 7772777; 7774777; 7778777

Câu 2. Từ đại diện (4 điểm)

Trong thao tác tìm kiếm các tập tin hoặc các từ trong một nội dung văn bản đã soạn thảo, ... người ta phải sử dụng một từ dùng làm từ khóa hay còn gọi là từ đại diện nhằm để so sánh và tìm kiếm kết quả. Một từ trong tin học được xem là một xâu kí tự không chứa kí tự trống.

Từ đại diện có thể chứa một vài kí tự dấu hỏi (?) mà mỗi kí tự dấu hỏi sẽ đại diện cho một kí tự bất kì khác kí tự trống (dấu cách). Chẳng hạn:

Từ đại diện	Ý nghĩa của từ đại diện
?zj?w	Từ đại diện là từ có độ dài bằng 5, có kí tự thứ nhất và thứ tư là các **kí tự bất kì** (khác kí tự trống), các kí tự thứ hai và ba là **z** và **j**, kí tự cuối cùng là **w**.

Trong phép so sánh, hai từ được xem là giống nhau nếu chúng có cùng số lượng các kí tự (độ dài của từ) và hai kí tự ở hai vị trí giống nhau của hai từ phải giống nhau hoặc trong hai kí tự này phải có ít nhất một kí tự là kí tự dấu hỏi.

Cho trước 2 xâu kí tự S và P, trong đó:

- Xâu S chứa nhiều từ, mỗi từ chỉ chứa các kí tự chữ cái thường hoặc chữ cái hoa và mỗi từ cách nhau ít nhất một kí tự trống. Xâu S có không quá 255 kí tự.
- Xâu P là một từ đại diện. Xâu P có không quá 255 kí tự.

Yêu cầu: Xác định có bao nhiêu từ giống từ P có trong xâu S (có phân biệt chữ thường và chữ hoa).

Dữ liệu: Đọc từ file văn bản **DAIDIEN.INP** hai xâu kí tự S và P như đã nêu ở trên, trong đó:

- Dòng đầu tiên chứa xâu S.
- Dòng thứ hai chứa xâu P.

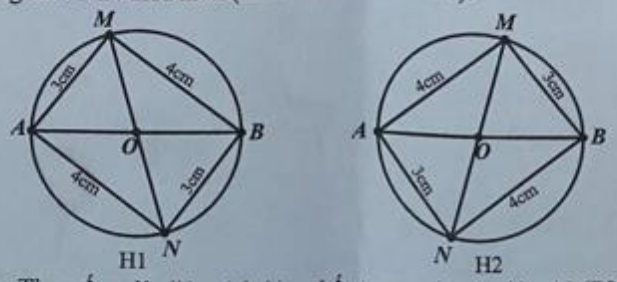
Kết quả: Ghi ra file văn bản **DAIDIEN.OUT** một số nguyên k là số từ giống từ P có trong xâu S.

Ví dụ:

DAIDIEN.INP	DAIDIEN.OUT
abc defk mjan abc Abc abc	2
tant xranz htjk hanr safd aann ?an?	3

Câu 3. Diện tích lớn nhất

Cho đường tròn (O) có đường kính AB dài k (cm) ($k \in N^*$). Ta vẽ các đường kính MN của đường tròn (O) sao cho các tứ giác AMBN có các cạnh đều là số nguyên *(với đơn vị cm)*. Chẳng hạn, với đường kính AB = 5cm thì ta có thể vẽ được hai đường kính MN thỏa mãn (hình minh họa ở dưới).



Yêu cầu: Tìm số m là diện tích lớn nhất trong các tứ giác AMBN ở trên *(đơn vị cm^2)*.

Dữ liệu: Đọc từ file văn bản **DIENTICH.INP** số nguyên dương k là độ dài đường kính AB *(đơn vị cm)* của đường tròn (O) đã nêu ở trên.

Giới hạn: $k < 10^8$

Kết quả: Ghi ra file văn bản **DIENTICH.OUT** số m theo yêu cầu của đề bài.

Chú ý: Trong trường hợp không thể vẽ được đường kính MN nào thỏa mãn tứ giác AMBN có các cạnh đều là số nguyên (với đơn vị cm) thì ghi số 0.

Ví dụ:

DIENTICH.INP	DIENTICH.OUT
5	12

---Hết---

BÀI GIẢI ĐỀ SỐ 9
(Học sinh giỏi 9 – Đà Nẵng – Năm học: 2020 – 2021)

1. Câu 1. Số nguyên tố cân bằng (3 điểm)

Câu 1. Số nguyên tố cân bằng (3 điểm)

Một số được gọi là số nguyên tố cân bằng nếu nó là số nguyên tố có 2k + 1 chữ số ($k \in N^*$), trong đó có 2k chữ số giống nhau và có đúng một chữ số ở vị trí chính giữa (tức vị trí thứ k + 1 tính từ trái sang phải) là khác với các chữ số còn lại. Chẳng hạn số 7778777 là số nguyên tố cân bằng.

Yêu cầu:

Nhập từ bàn phím một số nguyên dương k ($k \in N^*, k \le 7$) . Hãy tính và in ra màn hình số lượng các số nguyên tố cân bằng có 2k + 1 chữ số.

Ví dụ:

Nhập từ bàn phím	In ra màn hình
3	7

Giải thích: Có 7 số nguyên tố cân bằng có 7 chữ số: 1114111; 1117111; 3331333; 3337333; 7772777; 7774777; 7778777

1.1. Tóm tắt đề bài:

- Tìm tất cả **số nguyên tố cân bằng** có dạng **2k + 1** chữ số, trong đó:
  + Có **2k** chữ số **giống nhau** (chữ số a)
  + Có **1 chữ số ở giữa khác** (chữ số b)
  + Toàn bộ số là **số nguyên tố**.
- Ví dụ: nếu k = 3 thì số có 7 chữ số. Một số cân bằng ví dụ là 7778777 (6 chữ số 7, 1 chữ số giữa là 8).

1.2. Ý tưởng giải:

- Số cần tìm có dạng **aa…abaa…a** (bên trái và bên phải là các chữ số **a** có số lượng bằng nhau, chính giữa chỉ có **1** số **b** (b ≠ a), có tổng cộng **2k + 1** chữ số)
- Chữ số đầu tiên và cuối cùng luôn giống nhau (`a`), nên ta chỉ cần chọn hai chữ số:
  + `a`: chữ số lặp lại (0 - 9, nhưng không được để đầu là 0)
  + `b`: chữ số giữa (0 - 9, nhưng khác `a`)
- Từ cặp `(a, b)` và giá trị `k`, ta **tạo ra số cân bằng** theo công thức:
  so = str(a)*k + str(b) + str(a)*k (tức là **k** lần chữ số `a`, sau đó là `b`, rồi thêm **k** lần `a` nữa)
- Chuyển chuỗi **số cân bằng** vừa tạo sang số nguyên (**socanbang** = `int(so)`).
- **Kiểm tra xem số này có phải số nguyên tố không.** Nếu phải, thì **tăng biến đếm** lên 1 đơn vị
- In kết quả

1.3. Mô tả thuật toán:

- **Bước 1.** Tạo hàm kiểm tra số nguyên tố
- **Bước 2. Nhập** giá trị `k`.
- **Bước 3.** Khởi tạo biến `dem = 0` để đếm kết quả.
- **Bước 4. Duyệt tất cả** các chữ số `a` từ **0** đến **9**. Bỏ qua `a = 0` vì số không thể bắt đầu bằng **0**.
- **Bước 5.** Với mỗi `a`, **duyệt tất cả** các chữ số `b` từ **0** đến **9**. Bỏ qua trường hợp `a = b`.
- **Bước 6. Tạo số** có dạng `a...aba...a` theo công thức:
  so = str(a)*k + str(b) + str(a)*k
- **Bước 7.** Chuyển chuỗi **so** vừa tạo sang số nguyên (**socanbang** = `int(so)`).
- Bước 8. **Kiểm tra** số vừa chuyển, nếu là nguyên tố (`nguyento(socanbang)`) thì **tăng biến đếm lên 1 đơn vị**.
- **Bước 9.** In ra số lượng số tìm được (**dem**)

1.4. Code tham khảo (Python và C++):

1.4.1. Code Python:

```
import math # Khai báo thư viện toán học math, thường dùng khi chương trình có tính
căn bậc 2
# Tạo hàm kiểm tra số nguyên tố
def nguyento(x):
    # Nếu x <= 1 thì không phải số nguyên tố
    if x <= 1: return False
```

```
    # Nếu x là 2 hoặc 3 thì là số nguyên tố
    if x == 2 or x == 3: return True
    # Nếu x chia hết cho 2 hoặc 3 thì không phải số nguyên tố
    if x % 2 == 0 or x % 3 == 0: return False
    # Bắt đầu từ 5 (vì số nguyên tố > 3 có dạng 6k ± 1)
    i = 5
    # Kiểm tra đến căn bậc hai của x
    while i <= math.isqrt(x):
        # Nếu chia hết cho i hoặc i+2 thì không phải số nguyên tố
        if x % i == 0 or x % (i + 2) == 0:
            return False
        # Mỗi bước tăng 6 (kiểm tra cặp 6k-1 và 6k+1)
        i = i + 6
    # Nếu không chia hết cho số nào, x là số nguyên tố
    return True
fi = open('NGUYENTO.INP', 'r')  # Mở file để đọc
k = fi.read()                   # Đọc k từ file
k = int(k)                      # Chuyển số vừa đọc ra số nguyên
fi.close()                      # Đóng file sau khi đọc xong
dem = 0 # Khởi tạo biến  dem = 0 để đếm kết quả
# Duyệt tất cả các cặp chữ số (a, b)
for a in range(0, 10):       # Các chữ số a giống nhau
    for b in range(0, 10):   # Chữ số khác ở giữa b
        if a == b: continue # Bỏ qua trường hợp a = b
        so = str(a) * k + str(b) + str(a) * k    # Tạo số có dạng a...a b
a...a
        if so[0] == '0': continue                # Bỏ qua nếu có chữ số
đầu tiên là 0
        socanbang = int(so)                      # Chuyển chuỗi so vừa tạo
sang số nguyên
        if nguyento(socanbang): dem = dem + 1    # Nếu là nguyên tố
(nguyento(socanbang)) thì tăng biến đếm lên 1 đơn vị
fo = open('NGUYENTO.OUT', 'w')  # Mở file để ghi
fo.write(f'{dem}')              # Ghi kết quả ra file
fo.close()                      # Đóng file sau khi ghi
```

1.3.2. Code C++:

```
#include <iostream>
#include <bits/stdc++.h>     // Thu vien tong hop
#include <fstream>           // Khai bao cai nay de su dung ifstream va ofstream
#include <cmath>             // Khai bao cai nay de dung can bac 2
#include <string>            // Khai bao cai nay de xu li chuoi
using namespace std;
// Ham kiem tra so nguyen to
bool nguyento(long long x)
{
    // Neu x <= 1 thi khong phai so nguyen to
    if (x <= 1) return false;
```

```
    // Neu x la 2 hoac 3 thi la so nguyen to
    if (x == 2 || x == 3) return true;
    // Neu x chia het cho 2 hoac 3 thi khong phai so nguyen to
    if (x % 2 == 0 || x % 3 == 0) return false;
    // Bat dau kiem tra tu 5 (vi so nguyen to > 3 co dang 6k ± 1)
    long long i = 5;
    // Kiem tra den can bac hai cua x
    while (i <= sqrt(x))
    {
        // Neu x chia het cho i hoac i+2 thi khong phai so nguyen to
        if (x % i == 0 || x % (i + 2) == 0) return false;
        // Moi buoc tang 6 (kiem tra cap 6k-1 va 6k+1)
        i = i + 6;
    }
    // Neu khong chia het cho so nao, x la so nguyen to
    return true;
}
int main()
{
    ifstream fi("NGUYENTO.INP");                        // Mo file de doc
    int k;
    fi >> k;                                            // Doc k tu file
    fi.close();                                         // Dong file sau khi
doc
    int dem = 0;                                        // Khoi tao bien dem
= 0 de dem ket qua
    // Duyet tat ca cac cap chu so (a, b)
    for (int a = 0; a < 10; ++a)                        // Cac chu so a
giong nhau
    {
        for (int b = 0; b < 10; ++b)                    // Cac chu so b o
giua
        {
            if (a == b) continue;                       // Bo qua truong hop
a = b
            // Tao so co dang a...a b a...a
            string s = string(k, char('0' + a)) + char('0' + b) +
string(k, char('0' + a));
            if (s[0] == '0') continue;                  // Bo qua neu co chu
so dau tien la 0
            long long socanbang = stoll(s);             // Chuyen chuoi so
vua tao sang so nguyen
            if (nguyento(socanbang)) dem = dem + 1; // Neu la nguyen to
thi tang bien dem
        }
    }
    ofstream fo("NGUYENTO.OUT");                        // Mo file de ghi
```

```
    fo << dem;                                          // Ghi ket qua ra
file
    fo.close();                                         // Dong file sau khi
ghi
    return 0;
}
```

2. Câu 2. Từ đại diện (4 điểm)

Câu 2. Từ đại diện (4 điểm)

Trong thao tác tìm kiếm các tập tin hoặc các từ trong một nội dung văn bản đã soạn thảo, ... người ta phải sử dụng một từ dùng làm từ khóa hay còn gọi là từ đại diện nhằm để so sánh và tìm kiếm kết quả. Một từ trong tin học được xem là một xâu kí tự không chứa kí tự trống.

Từ đại diện có thể chứa một vài kí tự dấu hỏi (?) mà mỗi kí tự dấu hỏi sẽ đại diện cho một kí tự bất kì khác kí tự trống (dấu cách). Chẳng hạn:

Từ đại diện	Ý nghĩa của từ đại diện
?zj?w	Từ đại diện là từ có độ dài bằng 5, có kí tự thứ nhất và thứ tư là các **kí tự bất kì** (khác kí tự trống), các kí tự thứ hai và ba là **z** và **j**, kí tự cuối cùng là **w**.

Trong phép so sánh, hai từ được xem là giống nhau nếu chúng có cùng số lượng các kí tự (độ dài của từ) và hai kí tự ở hai vị trí giống nhau của hai từ phải giống nhau hoặc trong hai kí tự này phải có ít nhất một kí tự là kí tự dấu hỏi.

Cho trước 2 xâu kí tự S và P, trong đó:

- Xâu S chứa nhiều từ, mỗi từ chỉ chứa các kí tự chữ cái thường hoặc chữ cái hoa và mỗi từ cách nhau ít nhất một kí tự trống. Xâu S có không quá 255 kí tự.
- Xâu P là một từ đại diện. Xâu P có không quá 255 kí tự.

Yêu cầu: Xác định có bao nhiêu từ giống từ P có trong xâu S (có phân biệt chữ thường và chữ hoa).

Dữ liệu: Đọc từ file văn bản **DAIDIEN.INP** hai xâu kí tự S và P như đã nêu ở trên, trong đó:

- Dòng đầu tiên chứa xâu S.
- Dòng thứ hai chứa xâu P.

Kết quả: Ghi ra file văn bản **DAIDIEN.OUT** một số nguyên k là số từ giống từ P có trong xâu S.

Ví dụ:

DAIDIEN.INP	DAIDIEN.OUT
abc defk mjan abc Abc abc	2
tant xranz htjk hanr safd aann ?an?	3

2.1. Tóm tắt đề bài:

- **Cho hai dòng:**

  + Dòng 1: Chứa một chuỗi ký tự gồm nhiều từ, cách nhau bởi dấu cách - gọi là S.

  + Dòng 2: Chứa một từ mẫu P, có thể chứa ký tự đặc biệt '?'.

  + Ký tự '?' trong P có thể đại diện cho bất kỳ ký tự nào (ngoại trừ dấu cách).

- **Yêu cầu:**

  + Hãy đếm xem trong S có bao nhiêu từ khớp với mẫu P, trong đó:

    ◦ Hai từ phải có cùng độ dài.

    ◦ Mỗi vị trí ký tự trong từ phải trùng khớp hoàn toàn với P, hoặc ký tự tương ứng trong P là '?'.

    ◦ Có phân biệt chữ hoa và chữ thường.

- Kết quả ghi ra file **DAIDIEN.OUT**.

2.2. Ý tưởng giải:

- Dùng hàm `split()` để tách chuỗi S thành danh sách các từ.
- Duyệt qua từng từ trong danh sách đó.
- Với mỗi từ:

  + Nếu độ dài khác mẫu → Bỏ qua.

  + Ngược lại, kiểm tra từng ký tự:

    ▪ Nếu ký tự trong mẫu là `'?'` → Chấp nhận.

    ▪ Nếu khác `'?'` → Phải trùng chính xác với ký tự tương ứng trong từ.

  + Nếu tất cả vị trí đều đúng → Tăng biến đếm.

- Cuối cùng ghi biến đếm ra file.

2.3. Mô tả thuật toán:

- **Bước 1: Đọc dữ liệu từ file** `DAIDIEN.INP`**.**

  + Dòng 1 → Chuỗi S.

  + Dòng 2 → Từ mẫu P.

- **Bước 2: Tách các từ trong** S **thành danh sách các từ bằng lệnh:** danhsachtu = S.split()

- **Bước 3:**

  Khởi tạo biến `dem = 0` (đếm số từ khớp).

- **Bước 4:**

  + Với mỗi `tu` trong `danhsachtu`:

    - Nếu `len(tu) != len(P)` → Bỏ qua.
    - Ngược lại:
      - Gán `khop = True`.
      - Duyệt từng vị trí `i` từ `0` → `len(P)-1`:
        - Nếu `P[i] != '?'` và `P[i] != tu[i]`
          → Đặt `khop = False`, dừng vòng lặp.
      - Sau khi duyệt, nếu `khop == True` → tăng `dem` thêm 1.

- **Bước 5: Ghi kết quả `dem` vào file** `DAIDIEN.OUT`**.**

2.4. Code tham khảo (Python và C++):

2.4.1. Code Python:

```
# Đọc dữ liệu từ file DAIDIEN.INP
fi = open('DAIDIEN.INP', 'r')    # Mở file để đọc
S = fi.readline().strip()        # Đọc S
P = fi.readline().strip()        # Đọc P
fi.close()                       # Đóng file sau khi đọc xong
# Tách chuỗi S thành danh sách các từ riêng lẻ
danhsachtu = S.split()
dem = 0 # Khởi tạo biến dem = 0 (đếm số từ khớp)
# Với mỗi tu trong danhsachtu
for tu in danhsachtu:
    if len(tu) != len(P): continue  # Bỏ qua nếu độ dài khác
    khop = True # Giả sử ban đầu là khớp
    for i in range(0, len(P)):  # Duyệt từng vị trí i từ 0 → len(P)-1
        if P[i] != '?' and P[i] != tu[i]:
            khop = False
            break            # Chỉ cần sai 1 ký tự là dừng luôn
    if khop == True: dem = dem + 1  # Sau khi duyệt, nếu khop == True → tăng
dem thêm 1
# Ghi kết quả ra file
fo = open('DAIDIEN.OUT', 'w')    # Mở file để ghi
fo.write(f'{dem}')               # Ghi kết quả ra file
fo.close()                       # Đóng file sau khi ghi xong
```

2.4.2. Code C++:

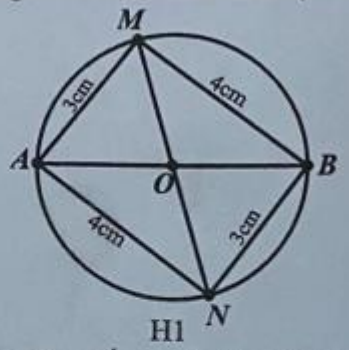
```
#include <iostream>
#include <bits/stdc++.h>     // Thu vien tong hop
#include <fstream>           // Khai bao cai nay de doc, ghi file
#include <sstream>           // Khai bao cai nay de de tach tung tu trong xau
#include <vector>            // Khai bao cai nay de dung duoc vector
#include <string>            // Khai bao cai nay de xu li xau
using namespace std;
int main()
{
    // Doc du lieu tu file DAIDIEN.INP
    ifstream fi("DAIDIEN.INP"); // Mo file de doc
```

```
    string S, P;
    getline(fi, S);                 // Doc S
    getline(fi, P);                 // Doc P
    fi.close();                     // Dong file sau khi doc xong
    // Tach chuoi S thanh danh sach cac tu rieng le
    stringstream ss(S);
    vector <string> danhsachtu;
    string tu;
    while (ss >> tu)
    {
        danhsachtu.push_back(tu);
    }
    int dem = 0;     // Khoi tao bien dem = 0 (dem so tu khop)
    // Voi moi tu trong danhsachtu
    for (string tu : danhsachtu)
    {
        if (tu.length() != P.length()) continue; // Bo qua neu do dai
khac
        bool khop = true; // Gia su ban dau la khop
        for (int i = 0; i < P.length(); i++)     // Duyet tung vi tri i tu
0 -> len(P)-1
        {
            if (P[i] != '?' && P[i] != tu[i])
            {
                khop = false; // Chi can sai 1 ky tu la dung luon
                break;
            }
        }
        if (khop == true) dem = dem + 1; // Sau khi duyet, neu khop == True
-> tang dem them 1
    }
    // Ghi ket qua ra file
    ofstream fo("DAIDIEN.OUT"); // Mo file de ghi
    fo << dem;                  // Ghi ket qua ra file
    fo.close();                 // Dong file sau khi ghi xong
    return 0;
}
```

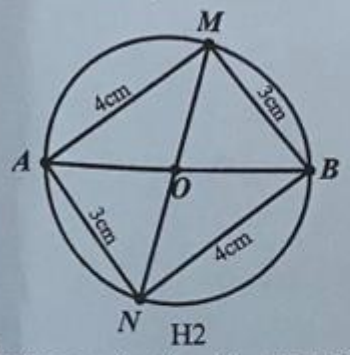
3. Câu 3. Diện tích lớn nhất

Câu 3. Diện tích lớn nhất

Cho đường tròn (O) có đường kính AB dài k (cm) ($k \in N^*$). Ta vẽ các đường kính MN của đường tròn (O) sao cho các tứ giác AMBN có các cạnh đều là số nguyên *(với đơn vị cm)*. Chẳng hạn, với đường kính AB = 5cm thì ta có thể vẽ được hai đường kính MN thỏa mãn (hình minh họa ở dưới).



H1



H2

Yêu cầu: Tìm số m là diện tích lớn nhất trong các tứ giác AMBN ở trên *(đơn vị cm^2)*.

Dữ liệu: Đọc từ file văn bản **DIENTICH.INP** số nguyên dương k là độ dài đường kính AB *(đơn vị cm)* của đường tròn (O) đã nêu ở trên.

Giới hạn: $k < 10^8$

Kết quả: Ghi ra file văn bản **DIENTICH.OUT** số m theo yêu cầu của đề bài.

Chú ý: Trong trường hợp không thể vẽ được đường kính MN nào thỏa mãn tứ giác AMBN có các cạnh đều là số nguyên (với đơn vị cm) thì ghi số 0.

Ví dụ:

DIENTICH.INP	DIENTICH.OUT
5	12

3.1. Tóm tắt đề bài:

- Cho đường tròn (O) có **đường kính AB = k** (với k là số nguyên dương).
- Trên cùng đường tròn, ta cần tìm một đường kính khác **MN** sao cho tứ giác **AMBN** có **các cạnh đều là số nguyên (cm)**.
- **Yêu cầu:** Tìm **diện tích lớn nhất** của tứ giác **AMBN** (đơn vị cm^2).
- Nếu **không có** tứ giác nào thỏa mãn điều kiện cạnh nguyên, in ra **0**.
- Ví dụ: Với **k = 5**, kết quả là **12**.

3.2. Ý tưởng giải:

- **Ta có:** Bốn đỉnh được tạo bởi hai đường kính cắt nhau trên cùng một đường tròn **luôn tạo thành một hình chữ nhật** - bất kể góc giữa hai đường kính là bao nhiêu. Do vậy trong bài này ta sử dụng định lý Pitago để giải.

=> Đề bài này trở thành: Tìm các **cặp số nguyên dương (a, b)** sao cho $a^2 + b^2 = k^2$, và chọn cặp cho S = a * b lớn nhất.

3.3. Mô tả thuật toán:

- **Bước 1. Nhập k** (độ dài đường kính AB).
- **Bước 2. Duyệt** tất cả các giá trị **a** từ **1** đến **k − 1**:
  + Tính $b = \sqrt{k^2 - a^2}$.
  + Nếu ***b*** là số nguyên → tính diện tích $S = a \times b$.
- **Bước 3.** Giữ giá trị **S lớn nhất** trong các trường hợp hợp lệ.
- **Bước 4.** Nếu không có cặp nào hợp lệ → ghi ra **0**.
- **Bước 5.** Ghi ra diện tích lớn nhất.

3.4. Code tham khảo (Python và C++):

3.4.1. Code Python:

```python
import math # Khai bao thu vien nay de tinh duoc can bac 2
fi = open('DIENTICH.INP', 'r')  # Mo file de doc
k = fi.read()                   # Doc k tu file
k = int(k)                      # Chuyen k vua doc sang so nguyen
fi.close()                      # Dong file sau khi doc xong
Smax = 0     # Khoi tao Smax = 0, dung de giu lai gia tri S lon nhat
for a in range(1, k):           # Duyet tat ca cac gia tri a tu 1 den k - 1
    b2 = k * k - a * a          # Tinh b binh phuong
    b = int(math.isqrt(b2))     # Tinh b
    if b * b == b2:             # Neu b la so nguyen
        S = a * b               # Tinh S
        if S > Smax: Smax = S   # So sanh S voi Smax de giu lai gia tri S lon
nhat
fo = open('DIENTICH.OUT', 'w')  # Mo file de ghi
fo.write(f'{Smax}')             # Ghi ket qua ra file
fo.close()                      # Dong file sau khi ghi xong
```

3.4.2. Code C++:

```cpp
#include <iostream>
#include <bits/stdc++.h>     // Thu vien tong hop
#include <fstream>           // Khai bao cai nay de doc, ghi file
#include <cmath>             // Khai bao cai nay de tinh can bac 2
using namespace std;
// Ham tinh can bac 2 nguyen
long long isqrt(long long n)
{
    long long x = sqrt((long double)n); // Lay gan dung
    while (x * x > n) x--;              // Dieu chinh xuong neu du
    while ((x + 1) * (x + 1) <= n) x++; // Dieu chinh len neu thieu
    return x;
}
```

```
int main()
{
    ifstream fi("DIENTICH.INP");     // Mo file de doc
    long long k;
    fi >> k;                         // Doc k tu file
    fi.close();                      // Dong file sau khi doc xong
    long long Smax = 0; // Khoi tao Smax = 0, dung de giu lai gia tri S lon nhat
    for (long long a = 1; a < k; a++)      // Duyet tat ca cac gia tri a tu
1 den k - 1
    {
        long long b2 = k * k - a * a;    // Tinh b binh phuong
        long long b = isqrt(b2);         // Tinh b
        if (b * b == b2)                 // Neu b la so nguyen
        {
            long long S = a * b;         // Tinh S
            if (S > Smax) Smax = S; // So sanh S voi Smax de giu lai gia tri
S lon nhat
        }
    }
    ofstream fo("DIENTICH.OUT");     // Mo file de ghi
    fo << Smax;                      // Ghi ket qua ra file
    fo.close();                      // Dong file sau khi ghi xong
    return 0;
}
```

4. Các bộ Test:

Mỗi câu của đề thi đều có đủ 10 bộ Test, với các giá trị đầu vào tương ứng với tỉ lệ (%) của các ràng buộc (Substack) mà đề bài yêu cầu. Những bộ Test này đã được gửi ở tệp tin đính kèm cùng với tài liệu này.

5. Kết quả chấm thử bằng phần mềm Themis 1.9, giới hạn thời gian chạy của mỗi bộ Test là 01 giây:

Tên bài	Điểm tối đa	Điểm chấm thử bằng Themis 1.9	
		Python	C++
Câu 1. Số nguyên tố cân bằng (NGUYENTO.*)	3.0 điểm	2.7 điểm (90%)	3.0 điểm (100%)
Câu 2. Từ đại diện (DAIDIEN.*)	4.0 điểm	4.0 điểm (100%)	4.0 điểm (100%)
Câu 3. Diện tích lớn nhất (DIENTICH.*)	3.0 điểm	1.5 điểm (50%)	2.4 điểm (80%)
Tổng số điểm đạt được:		**8.2 điểm (82%)**	**9.4 điểm (94%)**

ĐỀ SỐ 10
(Tin học trẻ Bảng B (THCS) – Đà Nẵng – Năm học: 2020 – 2021)

UBND THÀNH PHỐ ĐÀ NẴNG
HỘI THI TIN HỌC TRẺ CẤP THÀNH PHỐ
NĂM HỌC 2020-2021

ĐỀ THI BẢNG B – CẤP THCS
Thời gian: 120 phút (không kể thời gian giao đề)
(Đề thi có 3 trang)

ĐỀ CHÍNH THỨC

TỔNG QUAN

	Tên bài	File chương trình	File dữ liệu vào	File dữ liệu ra
Câu 1	Số giàu có	GIAUCO.*	Nhập từ bàn phím	In ra màn hình
Câu 2	Dịch cúm	CORONA.*	CORONA.INP	CORONA.OUT
Câu 3	Cắt dây	CATDAY.*	CATDAY.INP	CATDAY.OUT
Câu 4	Sắp xếp theo modul K	MODK.*	MODK.INP	MODK.OUT

*Chú ý: Dấu * có thể là pas; c; cpp hoặc py tương ứng với ngôn ngữ lập trình là Free Pacal; C/C++ hoặc Python.*

Câu 1. (2,0 điểm) Số giàu có

Trong các số tự nhiên từ 1 đến N, số tự nhiên được gọi là số giàu có nhất nếu nó có tổng các ước lớn nhất trong các số này.

Ví dụ: Số 12 là số giàu có nhất trong các số tự nhiên từ 1 đến 15. (Tổng ước của 12 là 1+2+3+4+6+12 = 28).

Yêu cầu: Hãy xác định số giàu có nhất trong các số tự nhiên từ 1 đến N.

Dữ liệu vào: Nhập từ bàn phím một số tự nhiên N ($0 < N \le 10^6$)

Dữ liệu ra: In ra màn hình số giàu có nhất trong các số tự nhiên từ 1 đến N.

Chú ý: Nếu kết quả có nhiều hơn một số thì in ra số nhỏ nhất trong các số đó.

Ví dụ:

Nhập từ bàn phím	In ra màn hình
15	12

Giới hạn: Có 80% test ứng với $N < 10^3$; 20% test ứng với $10^3 \le N \le 10^6$.

Câu 2. (2,0 điểm) Dịch cúm

Như chúng ta đã biết dịch cúm toàn cầu COVID-19 do virus Corona nhân bản và lây lan gây hội chứng suy hô hấp cấp tính nặng ở người. Người bệnh ban đầu không nhận biết được đã nhiễm bệnh do virus còn tiềm ẩn chưa khởi phát. Ở đâu đó những con virus đang ẩn mình, chúng ta cùng tìm chúng nhé!

Cho một xâu kí tự S chỉ chứa các kí tự C, O, R, N, A ở vị trí bất kì. Ta có thể thực hiện hoán đổi các kí tự này để tạo thành những cụm từ CORONA liên tiếp, mỗi cụm từ CORONA tương ứng với một con virus.

Ví dụ: Với xâu kí tự S = 'COOCROONRANNA', sau khi thực hiện hoán đổi các kí tự của xâu S ta được xâu 'CORONACORONAN' có hai cụm từ CORONA tương ứng với hai con virus.

Yêu cầu: Hãy xác định số lượng con virus Corona sau khi thực hiện hoán đổi các kí tự trong xâu S theo yêu cầu như trên.

Dữ liệu vào: Đọc từ file văn bản có tên **CORONA.INP** một xâu S chỉ chứa các kí tự C, O, R, N, A và có độ dài L ($0 < L \le 255$).

Dữ liệu ra: Ghi ra file văn bản **CORONA.OUT** một số nguyên là số lượng con virus Corana tạo ra sau khi hoán đổi các kí tự trong xâu S.

Ví dụ:

CORONA.INP	CORONA.OUT
COOCROONRANNA	2

Câu 3. Cắt dây (3,0 điểm)

Tý muốn cắt một sợi dây có chiều dài N (mét) thành 3 đoạn dây có chiều dài mỗi đoạn là số nguyên dương (đơn vị mét) sao cho 3 đoạn dây này là 3 cạnh của một tam giác cân có cạnh đáy lớn hơn cạnh bên.

Lưu ý: Tam giác cân là tam giác có hai cạnh bằng nhau, hai cạnh bằng nhau gọi là hai cạnh bên, cạnh còn lại gọi là cạnh đáy.

Yêu cầu: Em hãy giúp Tý tính có bao nhiêu cách cắt đoạn dây này.

Dữ liệu vào: Đọc ở file văn bản **CATDAY.INP** số nguyên dương N.

Giới hạn: $N \le 10^{16}$

Dữ liệu ra: Ghi ra file văn bản **CATDAY.OUT** số M là số cách cắt sợi dây theo yêu cầu.

Ví dụ:

CATDAY.INP	CATDAY.OUT
19	2

Giải thích:

Có 2 cách cắt sợi dây thành 3 đoạn thỏa mãn đề là: (5m; 5m; 9m) và (6m; 6m; 7m).

Lưu ý:

- *Các cách cắt sợi dây thành 3 đoạn (x mét; x mét; y mét) và các hoán vị của bộ 3 số (x;x;y) chỉ được tính là 1 cách cắt. Chẳng hạn: Cách cắt thành các đoạn (5m; 5m; 9m) và các hoán vị của nó là (5m; 9m; 5m) hoặc (9m; 5m; 5m) chỉ được tính là 1 cách cắt.*
- *Có 20% test ứng với $N \le 10^2$; 30% test ứng với $10^2 < N \le 10^6$; 30% test ứng với $10^6 < N \le 10^8$; 20% test ứng với $10^8 < N \le 10^{16}$.*

Câu 4. (3,0 điểm) Sắp xếp theo modul K

Từ dãy số tự nhiên 1; 2; 3; ...; N người ta sắp xếp lại dãy số này theo số dư trong các phép chia các số hạng của dãy số cho một số tự nhiên K là ước nào đó của N như sau:

- Đoạn thứ nhất gồm tất cả các số chia hết cho K.
- Đoạn thứ hai gồm tất cả các số chia K dư 1.
- Đoạn thứ ba gồm tất cả các số chia K dư 2.
- ....
- Đoạn cuối cùng gồm tất cả các số chia K dư K – 1.

Các số hạng trong mỗi đoạn cũng được sắp xếp theo chiều tăng dần.

Ví dụ: Với N = 12 và K = 4 sau khi sắp xếp ta có dãy số sau:

4; 8; 12; 1; 5; 9; 2; 6; 10; 3; 7; 11

Cho trước 3 số nguyên dương N; K; M (với K là ước của N và M ≤ N).

Yêu cầu: Tìm số hạng thứ M của dãy đã sắp xếp.

Dữ liệu vào: Đọc từ file văn bản **MODK.INP** 3 số nguyên dương N; K; M trên cùng một dòng, mỗi số cách nhau một dấu cách.

Giới hạn: $N \le 10^{16}$; $K \le 10^8$; K là ước của N; $M \le N$.

Dữ liệu ra: Ghi ra file văn bản **MODK.OUT** số hạng thứ M của dãy số theo yêu cầu.

Ví dụ:

MODK.INP	MODK.OUT
12 4 6	9

Lưu ý: Có 20% test ứng với $N \le 10^2$; 30% test ứng với $10^2 < N \le 10^6$; 30% test ứng với $10^6 < N \le 10^9$; 20% test ứng với $10^9 < N \le 10^{16}$.

-----Hết-----

BÀI GIẢI ĐỀ SỐ 10
(Tin học trẻ Bảng B (THCS) – Đà Nẵng – Năm học: 2020 – 2021)

1. Câu 1. (2,0 điểm) Số giàu có

Câu 1. (2,0 điểm) Số giàu có

Trong các số tự nhiên từ 1 đến N, số tự nhiên được gọi là số giàu có nhất nếu nó có tổng các ước lớn nhất trong các số này.

Ví dụ: Số 12 là số giàu có nhất trong các số tự nhiên từ 1 đến 15. (Tổng ước của 12 là 1+2+3+4+6+12 = 28).

Yêu cầu: Hãy xác định số giàu có nhất trong các số tự nhiên từ 1 đến N.

Dữ liệu vào: Nhập từ bàn phím một số tự nhiên N ($0 < N \le 10^6$)

Dữ liệu ra: In ra màn hình số giàu có nhất trong các số tự nhiên từ 1 đến N.

Chú ý: Nếu kết quả có nhiều hơn một số thì in ra số nhỏ nhất trong các số đó.

Ví dụ:

Nhập từ bàn phím	In ra màn hình
15	12

Giới hạn: Có 80% test ứng với $N < 10^5$; 20% test ứng với $10^5 \le N \le 10^6$.

1.1. Tóm tắt đề bài:

- Cho số nguyên dương **N ($0 < N \le 10^6$)**.
- Trong các số từ **1 đến N**, ta cần tìm **số giàu có** nhất - tức là số có **tổng các ước số** (bao gồm cả 1 và chính nó) **lớn nhất**.
- Nếu có nhiều số cùng đạt tổng ước lớn nhất thì **lấy số nhỏ nhất** trong các số đó.

1.2. Ý tưởng giải:

- Với mỗi số **i** từ **1** đến **N**, ta cần tính tổng các ước của **i**. Nếu ta duyệt từ **1** đến **i** để kiểm tra ước, độ phức tạp sẽ là: **O(N* căn bậc 2 của N)** => Với **N = 10^6** sẽ **không chạy kịp**.
- Cách tối ưu: **Dùng kỹ thuật giống** Sieve (sàng)
  + Khởi tạo mảng tổng các ước từ **1** đến **N** (**sum_uoc[i] = tổng các ước của i**)
  + Với mỗi số **i từ 1 đến N**:
    Ta cộng **i** vào tất cả các bội số của nó (Các bội số **j = i, 2i, 3i, ... ≤ N)**:
    `sum_uoc[j] = sum_uoc[j] + i` (vì **i** là ước của **j**)
  + Sau đó duyệt `sum_uoc từ 1 đến N` để tìm:
    ▪ Số có Tổng ước lớn nhất
    ▪ Nếu nhiều số cùng tổng, lấy số nhỏ nhất.
- **Giải thích vì sao phải: *"Cộng i vào tất cả các bội số của nó"*:**
  + Giả sử ta đang duyệt **i = 3**.
  + Các **số có 3 là ước** là: 3, 6, 9, 12, 15, 18, ...
  + Nói cách khác: **i là ước của mọi bội số j = i, 2i, 3i, ...**
  => Vì vậy, khi ta muốn tính **tổng các ước số** của từng số, chỉ cần:
    + Đến số **i**
    + Thêm **i** vào tổng ước của tất cả các số **j** mà **i là ước của j** (tức là các số **j = i, 2i, 3i, …**)

1.3. Mô tả thuật toán:

- **Bước 1.** Đọc số nguyên dương **N**.
- **Bước 2.** Khởi tạo mảng tổng các ước từ **1** đến **N** (**tong_uoc[i] = tổng các ước của i**) và ban đầu đều gán các giá trị là **0**
- **Bước 3.** Dùng vòng lặp để cộng các ước theo dạng sàng:
  + Với mỗi số **i từ 1 đến N**:
    Ta cộng **i** vào tất cả các bội số của nó (Các bội số **j = i, 2i, 3i, ... ≤ N)**:
    `tong_uoc[j] = tong_uoc[j] + i` (vì **i** là ước của **j**)
- **Bước 4.** Tạo các biến:
  + Biến lưu tổng ước lớn nhất tìm được:
    tong_lon_nhat = -1
  + Biến lưu số giàu có nhất (nếu bằng nhau về tổng ước thì lấy số nhỏ nhất):
    so_giau_co = -1

- **Bước 5.** Duyệt **i** từ **1** đến **N** để tìm số có tổng ước lớn nhất:
  + Nếu `tong_uoc[i] > tong_lon_nhat` hiện tại:
    ▪ Cập nhật `tong_lon_nhat = tong_uoc[i]`
    ▪ Cập nhật `so_giau_co = i`
  + Nếu `tong_uoc[i] = tong_lon_nhat` và `i < so_giau_co` thì:
    ▪ Cập nhật lại `so_giau_co = i`
- **Bước 6.** In ra `so_giau_co` cuối cùng.

1.4. Code tham khảo (Python và C++):

1.4.1. Code Python:

```python
# Đọc số nguyên dương N
fi = open('GIAUCO.INP', 'r')      # Mở file để đọc
N = int(fi.read().strip())        # Đọc N từ file
fi.close()                        # Đóng file sau khi đọc xong
# Khởi tạo mảng tổng các ước từ 1 đến N (tong_uoc[i] = tổng các ước của i) và ban đầu
đều gán các giá trị là 0
tong_uoc = [0] * (N + 1)
# Sử dụng kỹ thuật sàng để cộng các ước cho mỗi số
# Với mỗi số i, ta cộng i vào tất cả các bội số của nó
for i in range(1, N + 1):
    for j in range(i, N + 1, i):
        tong_uoc[j] = tong_uoc[j] + i
# Tạo biến lưu tổng ước lớn nhất tìm được
tong_lon_nhat = -1
# Tạo biến lưu số giàu có nhất (nếu bằng nhau về tổng ước thì lấy số nhỏ nhất)
so_giau_co = -1
# Duyệt qua các số từ 1 đến N để tìm số có tổng ước lớn nhất
for i in range(1, N + 1):
    # Nếu tổng ước của i lớn hơn tổng lớn nhất hiện tại thì:
    if tong_uoc[i] > tong_lon_nhat:
        tong_lon_nhat = tong_uoc[i] # Cập nhật tong_lon_nhat = tong_uoc[i]
        so_giau_co = i              # Cập nhật so_giau_co = i
    # Nếu tong_uoc[i] = tong_lon_nhat và i < so_giau_co thì:
    elif tong_uoc[i] == tong_lon_nhat and i < so_giau_co:
        so_giau_co = i  # Cập nhật lại so_giau_co = i
# In kết quả (In ra so_giau_co cuối cùng)
fo = open('GIAUCO.OUT', 'w')      # Mở file để ghi
fo.write(f'{so_giau_co}')         # Ghi kết quả ra file
fo.close()                        # Đóng file sau khi ghi xong
```

1.4.2. Code C++:

```cpp
#include <iostream>          // Thu vien nhap, xuat du lieu
#include <bits/stdc++.h>     // Thu vien tong hop
#include <fstream>           // Thu vien doc, ghi file
using namespace std;
```

```
int main()
{
    // Doc so nguyen duong N
    ifstream fi("GIAUCO.INP");  // Mo file de doc
    long long N;
    fi >> N;                    // Doc N tu file
    fi.close();                 // Dong file sau khi doc xong
    // Khoi tao mang tong cac uoc tu 1 den N (tong_uoc[i] = tong cac uoc cua i) va ban
dau deu gan cac gia tri la 0
    vector<long long> tong_uoc(N + 1, 0);
    // Su dung ky thuat sang de cong cac uoc cho moi so
    // Voi moi so i, ta cong i vao tat ca cac boi so cua no
    for (long long i = 1; i <= N; i++)
    {
        for (long long j = i; j <= N; j = j + i)
        {
            tong_uoc[j] = tong_uoc[j] + i;
        }
    }
    // Tao bien luu tong uoc lon nhat tim duoc
    long long tong_lon_nhat = -1;
    // Tao bien luu so giau co nhat (neu bang nhau ve tong uoc thi lay so nho nhat)
    long long so_giau_co = -1;
    // Duyet qua cac so tu 1 den N de tim so co tong uoc lon nhat
    for (long long i = 1; i <= N; i++)
    {
        // Neu tong uoc cua i lon hon tong lon nhat hien tai thi:
        if (tong_uoc[i] > tong_lon_nhat)
        {
            tong_lon_nhat = tong_uoc[i];     // Cap nhat tong_lon_nhat =
tong_uoc[i]
            so_giau_co = i;                  // Cap nhat so_giau_co = i
        }
        // Neu tong_uoc[i] = tong_lon_nhat va i < so_giau_co thi:
        else if (tong_uoc[i] == tong_lon_nhat && i < so_giau_co)
        {
            so_giau_co = i; // Cap nhat lai so_giau_co = i
        }
    }
    // In ket qua (In ra so_giau_co cuoi cung)
    ofstream fo("GIAUCO.OUT");  // Mo file de ghi
    fo << so_giau_co;           // Ghi ket qua ra file
    fo.close();                 // Dong file sau khi ghi xong
    return 0;
}
```

2. Câu 2. (2,0 điểm) Dịch cúm

Câu 2. (2,0 điểm) Dịch cúm

Như chúng ta đã biết dịch cúm toàn cầu COVID-19 do virus Corona nhân bản và lây lan gây hội chứng suy hô hấp cấp tính nặng ở người. Người bệnh ban đầu không nhận biết được đã nhiễm bệnh do virus còn tiềm ẩn chưa khởi phát. Ở đâu đó những con virus đang ẩn mình, chúng ta cùng tìm chúng nhé!

Cho một xâu kí tự S chỉ chứa các kí tự C, O, R, N, A ở vị trí bất kì. Ta có thể thực hiện hoán đổi các kí tự này để tạo thành những cụm từ CORONA liên tiếp, mỗi cụm từ CORONA tương ứng với một con virus.

Ví dụ: Với xâu kí tự S = 'COOCROONRANNA', sau khi thực hiện hoán đổi các kí tự của xâu S ta được xâu 'CORONACORONAN' có hai cụm từ CORONA tương ứng với hai con virus.

Yêu cầu: Hãy xác định số lượng con virus Corona sau khi thực hiện hoán đổi các kí tự trong xâu S theo yêu cầu như trên.

Dữ liệu vào: Đọc từ file văn bản có tên **CORONA.INP** một xâu S chỉ chứa các kí tự C, O, R, N, A và có độ dài L ($0 < L \leq 255$).

Dữ liệu ra: Ghi ra file văn bản **CORONA.OUT** một số nguyên là số lượng con virus Corana tạo ra sau khi hoán đổi các kí tự trong xâu S.

Ví dụ:

CORONA.INP	CORONA.OUT
COOCROONRANNA	2

2.1. Tóm tắt đề bài:

- Cho một xâu **S** gồm các ký tự trong tập **{C, O, R, N, A}**.
- Ta được phép **hoán đổi vị trí các ký tự tuỳ ý**, miễn là sau khi sắp xếp lại, ta tạo được nhiều nhất số lượng **cụm "CORONA"** (đúng thứ tự C-O-R-O-N-A).
- Hãy xác định **số lượng lớn nhất các xâu "CORONA"** có thể tạo được từ S.
- **Ví dụ:**

S = COOCROONRANNA => Có thể tạo được **2** lần "CORONA".

2.2. Ý tưởng giải:

- Một xâu "CORONA" gồm:

Ký tự	C	**O**	R	N	A
Số lượng	1	**2**	1	1	1

- Vì ta được phép **hoán đổi tùy ý**, thứ tự không quan trọng => Chỉ cần đếm số lượng ký tự trong xâu **S** ban đầu.
- **Số cụm "CORONA" tối đa tạo được = min(Số lượng chữ C, Số lượng chữ O // 2 *(Vì cần 2 chữ O)*, Số lượng chữ R, Số lượng chữ N, Số lượng chữ A).**

2.3. Mô tả thuật toán:

- **Bước 1. Đọc xâu S.**
- **Bước 2. Đếm số lần xuất hiện của từng ký tự:**
  `dem_C, dem_O, dem_R, dem_N, dem_A.`
- **Bước 3. Tính số cụm CORONA tối đa tạo được:**
  `so_cum = min(dem_C, dem_O // 2, dem_R, dem_N, dem_A)`
- **Bước 4. Ghi kết quả ra file** CORONA.OUT.

2.4. Code tham khảo (Python và C++):

2.4.1. Code Python:

```python
# Bước 1. Đọc xâu S.
fi = open('CORONA.INP', 'r')     # Mở file để đọc
S = fi.read().strip()            # Đọc S từ file
fi.close()                       # Đóng file sau khi đọc xong
# Bước 2. Đếm số lần xuất hiện của từng ký tự:
dem_C = S.count('C')
dem_O = S.count('O')
dem_R = S.count('R')
dem_N = S.count('N')
dem_A = S.count('A')
# Bước 3. Tính số cụm CORONA tối đa tạo được
so_cum = min(dem_C, dem_O // 2, dem_R, dem_N, dem_A)   # Chữ O cần gấp đôi (2 chữ O)
# Bước 4. Ghi kết quả ra file CORONA.OUT
fo = open('CORONA.OUT', 'w')     # Mở file để ghi
fo.write(f'{so_cum}')            # Ghi kết quả ra file
fo.close()                       # Đóng file sau khi ghi xong
```

2.4.2. Code C++:

```cpp
#include <iostream>          // Thu vien nhap, xuat du lieu
#include <bits/stdc++.h>     // Thu vien tong hop
#include <fstream>           // Thu vien doc, ghi file
using namespace std;
int main()
{
    // Buoc 1. Doc xau S.
    ifstream fi("CORONA.INP");  // Mo file de doc
    string S;
    getline(fi, S);             // Doc S tu file
    fi.close();                 // Dong file sau khi doc xong
    // Buoc 2. Dem so lan xuat hien cua tung ky tu:
    int dem_C = 0, dem_O = 0, dem_R = 0, dem_N = 0, dem_A = 0;
    for (char c : S)
    {
        if (c == 'C') dem_C++;
        else if (c == 'O') dem_O++;
```

```
        else if (c == 'R') dem_R++;
        else if (c == 'N') dem_N++;
        else if (c == 'A') dem_A++;
    }
    // Buoc 3. Tinh so cum CORONA toi da tao duoc
    int so_cum = min({dem_C, dem_O / 2, dem_R, dem_N, dem_A});   //
Chu O can gap doi (2 chu O)
    // Buoc 4. Ghi ket qua ra file CORONA.OUT
    ofstream fo("CORONA.OUT");  // Mo file de ghi
    fo << so_cum;               // Ghi ket qua ra file
    fo.close();                 // Dong file sau khi ghi xong
    return 0;
}
```

3. Câu 3. Cắt dây (3,0 điểm)

Câu 3. Cắt dây (3,0 điểm)

Tý muốn cắt một sợi dây có chiều dài N (mét) thành 3 đoạn dây có chiều dài mỗi đoạn là số nguyên dương (đơn vị mét) sao cho 3 đoạn dây này là 3 cạnh của một tam giác cân có cạnh đáy lớn hơn cạnh bên.

Lưu ý: Tam giác cân là tam giác có hai cạnh bằng nhau, hai cạnh bằng nhau gọi là hai cạnh bên, cạnh còn lại gọi là cạnh đáy.

Yêu cầu: Em hãy giúp Tý tính có bao nhiêu cách cắt đoạn dây này.

Dữ liệu vào: Đọc ở file văn bản **CATDAY.INP** số nguyên dương N.

Giới hạn: $N \le 10^{16}$

Dữ liệu ra: Ghi ra file văn bản **CATDAY.OUT** số M là số cách cắt sợi dây theo yêu cầu.

Ví dụ:

CATDAY.INP	CATDAY.OUT
19	2

Giải thích:

Có 2 cách cắt sợi dây thành 3 đoạn thỏa mãn đề là: (5m; 5m; 9m) và (6m; 6m; 7m).

Lưu ý:

- *Các cách cắt sợi dây thành 3 đoạn (x mét; x mét; y mét) và các hoán vị của bộ 3 số (x;x;y) chỉ được tính là 1 cách cắt. Chẳng hạn: Cách cắt thành các đoạn (5m; 5m; 9m) và các hoán vị của nó là (5m; 9m; 5m) hoặc (9m; 5m; 5m) chỉ được tính là 1 cách cắt.*
- *Có 20% test ứng với $N \le 10^2$; 30% test ứng với $10^2 < N \le 10^6$; 30% test ứng với $10^6 < N \le 10^9$; 20% test ứng với $10^9 < N \le 10^{16}$.*

3.1. Tóm tắt đề bài:

- Ta có một sợi dây dài **N** ($N \le 10^{16}$).
- Cần cắt sợi dây thành **3 đoạn dây có độ dài nguyên dương** sao cho **3 đoạn đó tạo thành một tam giác cân**.
- Hỏi có **bao nhiêu cách** cắt dây như vậy (chỉ xét bộ số (**x, x, y**), **không tính các hoán vị**).

3.2. Ý tưởng giải:

- Ta cắt sợi dây N thành 3 đoạn có độ dài nguyên dương tạo thành tam giác cân:

+ Hai cạnh bằng nhau là **x**

+ Cạnh đáy là **y**

+ Tổng độ dài:

2x + y = N => y = N – 2x

- Để tạo thành **tam giác cân**, theo đề: Hai cạnh bằng nhau **x** gọi là cạnh bên, cạnh còn lại **y** gọi là cạnh đáy (cạnh đáy phải **lớn hơn** cạnh bên) => y > x

- Điều kiện tam giác (bất đẳng thức tam giác):

+ Tam giác cân có:

▪ Hai cạnh bên: x, x

▪ Cạnh đáy: y

▪ Điều kiện: 2x > y

▪ Thế y = N − 2x vào:

$$2x > N - 2x \Rightarrow 4x > N \Rightarrow x > \frac{N}{4}$$

+ Điều kiện đáy lớn hơn cạnh bên:

$$y > x \Rightarrow N - 2x > x \Rightarrow N > 3x \Rightarrow x < \frac{N}{3}$$

- Kết luận:

+ x phải thỏa:

$$\boxed{\frac{N}{4} < x < \frac{N}{3}}$$

+ Do x nguyên:

$$x_{\min} = \left\lfloor \frac{N}{4} \right\rfloor + 1$$

$$x_{\max} = \left\lfloor \frac{N-1}{3} \right\rfloor$$

- Số đáp án:

+ Nếu $x_{min} > x_{max}$ => Không có cách nào => M = 0

+ Ngược lại: $M = x_{max} - x_{min} + 1$

3.3. Mô tả thuật toán:

- Bước 1. **Đọc số nguyên dương N từ file CATDAY.INP.**

- Bước 2. Tính giá trị nhỏ nhất của x

+ Từ điều kiện tam giác:

$$x > \frac{N}{4}$$

+ Nên:

$$x_{\min} = \lfloor N/4 \rfloor + 1$$

- Bước 3. Tính giá trị lớn nhất của x

+ Từ điều kiện cạnh đáy lớn hơn cạnh bên:

$$x < \frac{N}{3}$$

+ Do đó:

$$x_{\max} = \lfloor (N-1)/3 \rfloor$$

- Bước 4. Kiểm tra phạm vi hợp lệ

Nếu $x_{min} > x_{max}$ => Kết quả **M = 0**.

- Bước 5. Tính số lượng nghiệm hợp lệ

$M = x_{max} - x_{min} + 1$

- Bước 6. **Ghi M vào CATDAY.OUT.**

3.4. Code tham khảo (Python và C++):

3.4.1. Code Python:

```python
# Bước 1. Đọc số nguyên dương N từ file CATDAY.INP.
fi = open('CATDAY.INP', 'r')     # Mở file để đọc
N = int(fi.read().strip())       # Đọc N từ file
fi.close()                       # Đóng file sau khi đọc xong
# Bước 2. Xác định giới hạn dưới cho x
x_min = N // 4 + 1
# Bước 3. Xác định giới hạn trên cho x (x < N/3)
# Vì x la so nguyen va x < N/3 => x <= floor((N-1)/3)
x_max = (N - 1) // 3
# Bước 4. Kiểm tra nếu không có giá trị x hợp lệ
if x_min > x_max:    # Nếu x_min > x_max => Không có cách cắt nào => Số cách M = 0
    M = 0
else:
    M = x_max - x_min + 1
# Bước 5. Ghi kết quả ra file CATDAY.OUT
fo = open('CATDAY.OUT', 'w')     # Mở file để ghi
fo.write(f'{M}')                 # Ghi kết quả ra file
fo.close()                       # Đóng file sau khi ghi xong
```

3.4.2. Code C++:

```cpp
#include <iostream>          // Thu vien nhap, xuat du lieu
#include <bits/stdc++.h>     // Thu vien tong hop
#include <fstream>           // Thu vien doc, ghi file
using namespace std;
int main()
{
    // Buoc 1. Doc so nguyen duong N tu file CATDAY.INP.
    ifstream fi("CATDAY.INP");  // Mo file de doc
    long long N;
```

```
    fi >> N;                        // Doc N tu file
    fi.close();                     // Dong file sau khi doc xong
    // Buoc 2. Xac dinh gioi han duoi cho x
    long long x_min = N / 4 + 1;
    // Buoc 3. Xac dinh gioi han tren cho x (x < N/3)
    // Vi x la so nguyen va x < N/3 => x <= floor((N-1)/3)
    long long x_max = (N - 1) / 3;
    // Buoc 4. Kiem tra neu khong co gia tri x hop le
    long long M;
    if (x_min > x_max)  // Neu x_min > x_max => Khong co cach cat nao => So cach
M = 0
    {
        M = 0;
    }
    else
    {
        M = x_max - x_min + 1;
    }
    // Buoc 5. Ghi ket qua ra file CATDAY.OUT
    ofstream fo("CATDAY.OUT");  // Mo file de ghi
    fo << M;                        // Ghi ket qua ra file
    fo.close();                     // Dong file sau khi ghi xong
    return 0;
}
```

4. Câu 4. (3,0 điểm) Sắp xếp theo modul K

Câu 4. (3,0 điểm) Sắp xếp theo modul K

Từ dãy số tự nhiên 1; 2; 3; ...; N người ta sắp xếp lại dãy số này theo số dư trong các phép chia các số hạng của dãy số cho một số tự nhiên K là ước nào đó của N như sau:

- Đoạn thứ nhất gồm tất cả các số chia hết cho K.
- Đoạn thứ hai gồm tất cả các số chia K dư 1.
- Đoạn thứ ba gồm tất cả các số chia K dư 2.
- .....
- Đoạn cuối cùng gồm tất cả các số chia K dư K – 1.

Các số hạng trong mỗi đoạn cũng được sắp xếp theo chiều tăng dần.

Ví dụ: Với N = 12 và K = 4 sau khi sắp xếp ta có dãy số sau:

4; 8; 12; 1; 5; 9; 2; 6; 10; 3; 7; 11

Cho trước 3 số nguyên dương N; K; M (với K là ước của N và M ≤ N).

Yêu cầu: Tìm số hạng thứ M của dãy đã sắp xếp.

Dữ liệu vào: Đọc từ file văn bản **MODK.INP** 3 số nguyên dương N; K; M trên cùng một dòng, mỗi số cách nhau một dấu cách.

Giới hạn: $N \le 10^{16}$; $K \le 10^{9}$; K là ước của N; $M \le N$.

Dữ liệu ra: Ghi ra file văn bản **MODK.OUT** số hạng thứ M của dãy số theo yêu cầu.

Ví dụ:

MODK.INP	MODK.OUT
12 4 6	9

Lưu ý: Có 20% test ứng với $N \le 10^2$; 30% test ứng với $10^2 < N \le 10^6$; 30% test ứng với $10^6 < N \le 10^9$; 20% test ứng với $10^9 < N \le 10^{16}$.

4.1. Tóm tắt đề bài:

- Đề bài yêu cầu sắp xếp N số nguyên dương liên tiếp từ 1 đến N thành một dãy dựa trên phần dư khi chia cho K. Dãy số được chia thành K đoạn liên tiếp:

+ **Đoạn 1 (nhóm i = 1):** Gồm tất cả các số chia hết cho K (phần dư là 0).

+ **Đoạn 2 (nhóm i = 2):** Gồm tất cả các số chia K dư 1.

+ **...**

+ **Đoạn K + 1 (nhóm i = K+1):** Gồm tất cả các số chia K dư K - 1.

- **Lưu ý quan trọng:** Đề bài nói có **K đoạn** liên tiếp: đoạn 1 (dư 0), đoạn 2 (dư 1), ..., đoạn K (dư K - 1). Tức là có K nhóm, và phần dư là từ 0 đến K - 1.

+ **Nhóm i = 1**: Phần dư là 0.

+ **Nhóm i = 2**: Phần dư là 1.

+ **...**

+ **Nhóm i = K**: Phần dư là K - 1.

- Trong mỗi đoạn (nhóm), các số được sắp xếp theo thứ tự **tăng dần**.
- Yêu cầu là tìm số hạng thứ M trong dãy số đã sắp xếp này.
- **Ví dụ:** N = 12, K = 4, M = 6.

+ **Dư 0 (Nhóm 1):** {4, 8, 12}

+ **Dư 1 (Nhóm 2):** {1, 5, 9}

+ **Dư 2 (Nhóm 3):** {2, 6, 10}

+ **Dư 3 (Nhóm 4):** {3, 7, 11}

+ Dãy sắp xếp: 4, 8, 12, **1**, 5, 9, **2**, 6, 10, **3**, 7, 11 => **Số hạng thứ 6** là 9.

4.2. Ý tưởng giải:

- Duyệt qua các nhóm **i** có **phần dư** từ **1** đến **K**.
- Tính số lượng phần tử (**sophantu_nhom_i**) trong mỗi nhóm **i**.
- Sử dụng phương pháp **cộng dồn** số phần tử (**sophantu_nhom_i**) vào tổng số phần tử đã có từ các nhóm trước đó (**tongsophantu**) để xác định nhóm chứa phần tử thứ **M**.
- Nếu **tổng số phần tử cộng dồn >= M** => Tính vị trí tương đối (**k**) của phần tử thứ **M** trong nhóm đó theo công thức:

k = M – sophantu_nhom_i

- Áp dụng công thức tính giá trị của phần tử (**X**) dựa trên vị trí (**k**) và nhóm (**i**):

+ **X = k * K**

+ **X = (i - 1) + (k - 1) * K**

4.3. Mô tả thuật toán:

- Bước 1. Đọc N, K, M từ file

- Bước 2. Khởi tạo các giá trị ban đầu

+ Khởi tạo biến lưu kết quả **X**. Nếu **M > N** (trái với ràng buộc), **X** sẽ in ra **-1**. (**X = -1**)

+ Khởi tạo tổng số phần tử của tất cả các nhóm (**tongsophantu**)

- Bước 3. Duyệt qua K nhóm i, tương ứng với i = 1 đến K

+ Tính số phần tử (**sophantu_nhom_i**) của từng nhóm **i**

▪ Trường hợp **i = 1** (Nhóm chia hết cho M, dư 0) **=> sophantu_nhom_i = N // K**

▪ Trường hợp **i > 1** (Nhóm chia M dư i-1):

◦ Khởi tạo số đầu tiên (**sodautien**) trong nhóm **i** (số nhỏ nhất >= 1 chia K dư i-1): **sodautien = i – 1**

◦ Nếu số đầu tiên đã lớn hơn **N** thì nhóm này không có phần tử nào. Ngược lại, số phần tử của nhóm này sẽ được tính theo công thức:

sophantu_nhom_i = (N - sodautien) // K + 1

- Bước 4. Kiểm tra M có nằm trong nhóm i hay không

+ Kiểm tra nếu tổng số phần tử của các nhóm trước (**tongsophantu**) cộng với số phần tử của nhóm hiện tại (**sophantu_nhom_i**) lớn hơn hoặc bằng **M**, thì phần tử thứ **M** phải nằm trong nhóm **i**.

+ Sau khi kiểm tra, nếu đúng là phần tử **M** có nằm trong nhóm **i** thì ta tiếp tục làm như sau:

▪ Tính vị trí tương đối **k** của phần tử cần tìm trong nhóm **i**: **k = M – tongsophantu**

▪ Tính kết quả **X** (Giá trị của phần tử thứ M):

◦ Trường hợp **i = 1** (Dư 0): **X = k * K**

◦ Trường hợp **i > 1** (Dư = i-1): **X = (i - 1) + (k - 1) * K**

◦ Thoát ra khỏi việc tính toán, vì đã tìm ra được kết quả

+ Trong trường hợp chưa tìm ra được kết quả thì tiếp tục xét đến đoạn kế tiếp:

tongsophantu = tongsophantu + sophantu_nhom_i

- Bước 5. Ghi kết quả đã tìm được ra file

* **<u>Giải thích cụ thể từng công thức có trong thuật toán</u>** *

1. <u>Công thức 1:</u> **sophantu_nhom_i = N // K**

- Công thức này dùng để tính số lượng các số chia hết cho K trong phạm vi từ 1 đến N.

- N: Là giới hạn trên (số lớn nhất) của dãy số [1, 2, .., N].

- K: Là số chia (ví dụ: nếu K = 4, ta tìm các bội số của 4).

- //: Là phép chia lấy phần nguyên. Phép toán này trả về phần nguyên của thương số.

- Khi thực hiện phép chia N / K, ta đang xem N lớn gấp bao nhiêu lần K. Phần nguyên của kết quả chính là số lần K thuộc về dãy N số đó, tương ứng với số lượng bội số của K nhỏ hơn hoặc bằng N.

- Minh họa bằng Dãy số:

+ Các số chia hết cho K trong khoảng [1, N] là các bội số của K: K * 1, K * 2, K * 3, .., K * k

+ Ta cần tìm giá trị k lớn nhất (k_{max}) sao cho: $K * k \leq N$

+ Chia cả hai vế cho K:

$k \leq N / K$

+ Vì k phải là số nguyên dương nên giá trị lớn nhất của k chính là phần nguyên của N / K:

$k_{max} = N // K$

+ Số lượng phần tử (sophantu_nhom_i) chính là k_{max}:

Sophantu_nhom_i = N // K

2. Công thức 2: **sodautien = i – 1**

- Đây là số hạng đầu tiên của cấp số cộng trong nhóm đó
- Khi i = 1 (nhóm dư 0), phần tử đầu tiên là K (được tính bằng công thức riêng N // K).
- Khi i > 1 (nhóm dư r = i-1): **sodautien** được định nghĩa là số nguyên dương nhỏ nhất ≥ 1 mà khi chia cho K sẽ cho phần dư là r = i-1.
- Vì r = i-1 là phần dư và 1 ≤ r ≤ K - 1, nên bản thân số r (tức là i-1) đã là số nhỏ nhất lớn hơn hoặc bằng 1 thỏa mãn điều kiện đó.
- Nói cách khác, (i - 1) tương đương với (i – 1) mod K
- Do đó, đối với các nhóm có phần dư r ≥ 1 (tức là i > 1), số hạng đầu tiên (**sodautien**) của nhóm đó chính là phần dư i - 1.
- Ví dụ minh họa: N = 10, K = 4 (Duyệt i chạy từ 1 đến 4)

i	**r = i – 1**	**sodautien = i – 1**	**Dãy số trong nhóm**
1			Đã tính bằng công thức riêng N // K
2	1	**1**	**1**, 5, 9, …
3	2	**2**	**2**, 6, 10, …
4	3	**3**	**3**, 7, …

3. Công thức 3: **sophantu_nhom_i = (N - sodautien) // K + 1**

- Công thức này áp dụng cho các nhóm i > 1 (các số chia K dư r ≥ 1).
- Mục đích là đếm số lượng phần tử k trong dãy số của nhóm i (dãy này là một cấp số cộng) mà thỏa mãn điều kiện $a_k \le N$.
- N: Giới hạn trên (số lớn nhất có thể).
- K: Công sai (khoảng cách giữa các số liên tiếp).
- sodautien: Số hạng đầu tiên của nhóm (chính là phần dư r = i-1).
- (N - sodautien): Tính toán khoảng cách từ số đầu tiên đến giới hạn N. Đây là phần còn lại của dãy số mà ta cần tìm số lần K có thể lọt vào.
- (N - sodautien) // K: Tính toán số lượng khoảng nhảy K (số lần công sai K lặp lại) sau số hạng đầu tiên.
- + 1: Cộng thêm 1 là để tính chính bản thân số hạng đầu tiên (sodautien) vào tổng số phần tử.
- Nói cách khác, công thức này là cách tính số lượng số hạng (k) của một cấp số cộng khi biết số hạng đầu (a_1), công sai (d = K), và giới hạn cuối ($a_k \le N$).

4. Công thức 4: **k = M – tongsophantu**

- Công thức này dùng để tìm vị trí tương đối của phần tử cần tìm (M) bên trong nhóm hiện tại, sau khi đã loại bỏ tất cả các phần tử của các nhóm đã được sắp xếp trước đó.
- M: Là số thứ tự tuyệt đối của phần tử cần tìm trong toàn bộ dãy số đã được sắp xếp (ví dụ: M = 8 nghĩa là tìm số hạng thứ 8).
- tongsophantu: Là tổng số phần tử của tất cả các nhóm đã được xét và đã được sắp xếp đứng trước nhóm hiện tại.
- k: Là vị trí tương đối (hay số thứ tự) của phần tử cần tìm bên trong nhóm hiện tại (ví dụ: k = 3 nghĩa là nó là phần tử thứ 3 trong nhóm này).

- Ví dụ minh họa:

+ Giả sử dãy số của bạn được sắp xếp thành các nhóm theo thứ tự: Nhóm 1, Nhóm 2, Nhóm 3, ...

+ Kịch bản 1: Tìm M = 8, giả sử ta đang xét Nhóm 3.

▪ Nhóm 1 có 2 phần tử: {4, 8}

▪ Nhóm 2 có 3 phần tử: {1, 5, 9}

▪ tongsophantu (tổng số phần tử của Nhóm 1 và Nhóm 2) = 2 + 3 = 5. Điều này có nghĩa là, 5 phần tử đầu tiên của dãy số nằm ở Nhóm 1 và Nhóm 2.

▪ Nhóm 3 (Dư 2) có 3 phần tử: {2, 6, 10}

◦ Ta biết rằng phần tử thứ M = 8 phải nằm trong Nhóm 3.

Vị trí thứ 6 là phần tử đầu tiên của Nhóm 3 (số 2).

Vị trí thứ 7 là phần tử thứ hai của Nhóm 3 (số 6).

Vị trí thứ 8 là phần tử thứ ba của Nhóm 3 (số 10).

+ Áp dụng Công thức:

▪ Để tìm vị trí tương đối k của phần tử thứ 8 trong Nhóm 3, ta lấy số thứ tự tuyệt đối trừ đi tổng số phần tử đã đi qua:

k = M – tongsophantu = 8 – 5 = 3 => **k = 3**

▪ Kết quả **k = 3** cho biết phần tử thứ 8 trong toàn bộ dãy chính là phần tử thứ 3 trong Nhóm 3.

5. <u>Công thức 5:</u> **X = k * K**

- Công thức X = k * K chỉ được áp dụng trong trường hợp khi tìm kiếm phần tử trong Nhóm 1 (i = 1), tức là nhóm gồm các số chia hết cho K (phần dư bằng 0).

- K: Số chia (Ví dụ: Nếu K = 4, ta tìm bội số của 4).

- k: Vị trí tương đối của phần tử cần tìm bên trong Nhóm 1. (Được tính từ công thức:
k = M – tongsophantu ở trên).

- X: Giá trị của phần tử cần tìm.

- Nhóm 1 bao gồm tất cả các số nguyên dương từ 1 đến N mà chia hết cho K. Đó chính là các bội số của K.

- Dãy số của Nhóm 1 được sắp xếp theo thứ tự tăng dần như sau: K * 1, K * 2, K * 3, K * 4, …

Vị trí tương đối k	**Giá trị phần tử X = k * K**
k = 1 (Phần tử thứ 1)	X = 1 * K
k = 2 (Phần tử thứ 2)	X = 2 * K
k = 3 (Phần tử thứ 3)	X = 3 * K
…	…
k = k (Phần tử thứ k)	X = k * K

- Vì vậy, nếu phần tử cần tìm là phần tử thứ k trong nhóm các số chia hết cho K, thì giá trị của nó chắc chắn là X = k * K.

6. <u>Công thức 6:</u> **X = (i - 1) + (k - 1) * K**

- Công thức này được áp dụng cho các trường hợp i > 1, tức là khi tìm kiếm phần tử trong các nhóm có phần dư r lớn hơn hoặc bằng 1.

- Đây là công thức cấp số cộng cơ bản: $a_k = a_1 + (k – 1) * d$
- Trong đó:

a_k: Là số hạng cần tìm (tương ứng với X)

a_1: Là số hạng đầu tiên (tương ứng với (i - 1))

d: Là công sai (khoảng cách giữa các số, tương ứng với K).

k: Là số thứ tự của số hạng trong nhóm

- Số hạng đầu tiên: **a_1 = (i – 1)**

+ i là số thứ tự của nhóm (i = 2, 3, …, K).

+ (i - 1) chính là phần dư r (r = 1, 2, …, K-1).

+ Đối với các nhóm có phần dư r ≥ 1, số nguyên dương nhỏ nhất thỏa mãn điều kiện đó chính là chính phần dư r đó.

▪ Ví dụ K = 4:

◦ Nhóm i = 2 (dư **1**): a_1 = 2 - 1 = **1**

◦ Nhóm i = 3 (dư **2**): a_1 = 3 - 1 = **2**

◦ Nhóm i = 4 (dư **3**): a_1 = 4 - 1 = **3**

▪ Vậy, (i - 1) đóng vai trò là số hạng đầu tiên (a_1) của nhóm.

- Công sai: **d = K**

+ Vì nhóm bao gồm các số có cùng phần dư khi chia cho K, nên khoảng cách giữa hai số liên tiếp trong nhóm luôn là K.

+ Ví dụ: 1, 5, 9, 13, … (Công sai 5 - 1 = 4, 9 - 5 = 4, … => K = 4).

+ Vậy, K đóng vai trò là công sai (d).

- Số thứ tự: **(k - 1)**

+ k là vị trí của phần tử trong nhóm (ví dụ: k = 3).

+ (k - 1) là số lần bước nhảy K tính từ số hạng đầu tiên.

- Ví dụ minh họa:

+ Giả sử N = 10, K = 4. Ta tìm phần tử thứ 3 (k = 3) trong Nhóm 3 (i = 3, dư 2).

▪ Số hạng đầu tiên (a_1): (i - 1) = 3 - 1 = 2.

▪ Công sai (d): K = 4.

▪ Vị trí (k): k = 3.

+ Các số trong Nhóm 3 là: 2, 6, 10.

+ Ta áp dụng công thức để tìm giá trị của phần tử thứ k = 3:

$X = a_1 + (k – 1) * d = 2 + (3 – 1) * 4 = 10$

+ Kết quả khớp với thực tế: Phần tử thứ 3 của nhóm là 10.

- Tóm lại, công thức X = (i - 1) + (k - 1) * K là sự chuyển đổi chính xác từ công thức cấp số cộng cơ bản sang ngôn ngữ của bài toán này.

7. <u>Công thức 7</u>: **tongsophantu = tongsophantu + sophantu_nhom_i**

- Công thức này là một phép tính cộng dồn (cumulative sum) hay còn gọi là tổng tiền tố (prefix sum), đóng vai trò then chốt trong việc xác định vị trí của phần tử thứ M.

- Mục đích của việc sử dụng công thức này là để theo dõi tổng số phần tử đã được sắp xếp vào dãy số cho đến hết nhóm hiện tại.

+ Mỗi khi vòng lặp chuyển sang xét một nhóm mới (i = 1, 2, 3, …), ta cần cộng số phần tử của nhóm đó vào tổng.

+ Giá trị cuối cùng của **tongsophantu** trước khi xét một nhóm mới cho biết phần tử cuối cùng của các nhóm trước đó nằm ở vị trí nào trong dãy số tổng thể.

- tongsophantu (trước khi cộng): Là số lượng phần tử của các nhóm đứng trước.

- tongsophantu + sophantu_nhom_i: Là tổng số phần tử nếu ta tính cả nhóm hiện tại.

- Nếu tổng này lớn hơn hoặc bằng M, điều đó có nghĩa là phần tử thứ M chắc chắn phải nằm trong nhóm hiện tại (i) và ta dừng vòng lặp.

- Ngược lại, nếu tổng nhỏ hơn M, ta cần cộng dồn và xét tiếp nhóm kế tiếp.

- Tóm lại, công thức **tongsophantu = tongsophantu + sophantu_nhom_i** đảm bảo rằng bạn biết được đã có bao nhiêu phần tử nằm trước vị trí bạn đang tìm kiếm, giúp xác định được chính xác nhóm chứa M.

4.4. Code tham khảo (Python và C++):

4.4.1. Code Python:

```
# Bước 1. Đọc N, K, M từ file
fi = open('MODK.INP', 'r')                # Mở file để đọc
N, K, M = map(int, fi.read().split())     # Đọc N, K, M từ file
fi.close()                                # Đóng file sau khi đọc xong
# Bước 2. Khởi tạo các giá trị ban đầu
X = -1  # Khởi tạo biến lưu kết quả X. Nếu M > N (trái với ràng buộc), X sẽ in ra -1.
tongsophantu = 0     # Khởi tạo Tổng số phần tử của tất cả các nhóm
# Bước 3. Duyệt qua M nhóm i, tương ứng với i = 1 đến K.
for i in range(1, K + 1):
    # - Tính số lượng phần tử của từng nhóm i(sophantu_nhom_i):
    #   + Trường hợp i = 1 (Nhóm chia hết cho K, dư 0):
    if i == 1:
        sophantu_nhom_i = N // K    # Đây là công thức để tính số lượng số
chia hết cho K trong khoảng [1, N].
    #   + Trường hợp i > 1 (Nhóm chia K dư i-1):
    else:
        sodautien = i - 1     # Đây là số đầu tiên trong nhóm i (số nhỏ nhất >=
1 chia K dư i-1)
        if sodautien > N:     # Nếu số đầu tiên đã lớn hơn N thì nhóm này không
có phần tử nào.
            sophantu_nhom_i = 0
        else:
            # Công thức này tính số lượng số chia K dư i-1 trong khoảng [1, N].
            sophantu_nhom_i = (N - sodautien) // K + 1
    # Bước 4. Kiểm tra M có nằm trong nhóm i hay không
```

```
    #       + Kiểm tra nếu tổng số phần tử của các nhóm trước (tongsophantu)
    #       cộng với số phần tử của nhóm hiện tại (sophantu_nhom_i) lớn hơn hoặc bằng
M,
    #       thì phần tử thứ M phải nằm trong nhóm i.
    if tongsophantu + sophantu_nhom_i >= M:
        # - Tính vị trí tương đối k của phần tử cần tìm trong nhóm i.
        k = M - tongsophantu
        # - Tính kết quả X (Giá trị của phần tử thứ M):
        #       + Trường hợp i = 1 (Dư 0):
        if i == 1:
            X = k * K    # Các số trong nhóm này là K, 2K, 3K,... => Phần tử thứ
k là k * K
        #       + Trường hợp i > 1 (Dư = i-1):
        #           Các số trong nhóm này là: (i-1), (i-1) + K, (i-1) + 2K, ... (Do
i-1 là số dư nhỏ nhất >= 1).
        #           Phần tử thứ k (ứng với k = 1, 2, 3, ...) sẽ là: (i-1) + (k-1) *
K.
        else:
            X = (i - 1) + (k - 1) * K
        break    # Thoát ra khỏi việc tính toán, vì đã tìm ra được kết quả
    # Trong trường hợp chưa tìm ra được kết quả thì tiếp tục xét đến đoạn kế tiếp
    tongsophantu = tongsophantu + sophantu_nhom_i
# Bước 5. Ghi kết quả đã tìm được ra file
fo = open('MODK.OUT', 'w')  # Mở file để ghi
fo.write(f'{X}')            # Ghi kết quả ra file
fo.close()                  # Đóng file sau khi ghi xong
```

4.4.2. Code C++:

```
#include <iostream>          // Thu vien nhap, xuat du lieu
#include <bits/stdc++.h>     // Thu vien tong hop
#include <fstream>           // Thu vien doc, ghi file
using namespace std;
int main()
{
    // Buoc 1. Doc N, K, M tu file
    ifstream fi("MODK.INP");     // Mo file de doc
    long long N, K, M;
    fi >> N >> K >> M;           // Doc N, K, M tu file
    fi.close();                  // Dong file sau khi doc xong
    // Buoc 2. Khoi tao cac gia tri ban dau
    long long X = -1;            // Khoi tao bien luu ket qua X. Neu M > N
(trai voi rang buoc), X se in ra -1.
    long long tongsophantu = 0; // Khoi tao Tong so phan tu cua tat ca cac
nhom
```

```
    // Buoc 3. Duyet qua K nhom, tuong ung voi i = 1 den K.
    for (long long i = 1; i <= K; i++)
    {
        // - Tinh so luong phan tu cua tung nhom i (sophantu_nhom_i)
        long long sophantu_nhom_i = 0;
        //      + Truong hop i = 1 (Nhom chia het cho K, du 0):
        if (i == 1)
        {
            sophantu_nhom_i = N / K;     // Day la cong thuc de tinh so
luong so chia het cho K trong khoang [1, N].
        }
        //      + Truong hop i > 1 (Nhom chia K du i-1):
        else
        {
            long long sodautien = i - 1;     // Day la so dau tien trong
nhom i (so nho nhat >= 1 chia K du i-1)
            if (sodautien > N)  // Neu so dau tien da lon hon N thi nhom nay
khong co phan tu nao.
            {
                sophantu_nhom_i = 0;
            }
            else
            {
                // Cong thuc nay tinh so luong so chia K du i-1 trong khoang [1,
N].
                sophantu_nhom_i = (N - sodautien) / K + 1;
            }
        }
        // Buoc 4. Kiem tra M co nam trong nhom i hay khong
        //      + Kiem tra neu tong so phan tu cua cac nhom truoc (tongsophantu)
        //      cong voi so phan tu cua nhom hien tai (sophantu_nhom_i) lon hon hoac
bang M,
        //      thi phan tu thu M phai nam trong nhom i.
        if (tongsophantu + sophantu_nhom_i >= M)
        {
            // - Tinh vi tri tuong doi k cua phan tu can tim trong nhom i.
            long long k = M - tongsophantu;
            // - Tinh ket qua X (Gia tri cua phan tu thu M):
            //      + Truong hop i = 1 (Du 0):
            if (i == 1)
            {
                // Cac so trong nhom nay la K, 2K, 3K, ..., => Phan tu thu k = k *
K
                X = k * K;
```

```
                }
                //      + Truong hop i > 1 (Du = i-1):
                //          Cac so trong nhom nay la: (i-1), (i-1) + K, (i-1) + 2K, ...
(Do i-1 la so du nho nhat >= 1).
                //          Phan tu thu k (ung voi k = 1, 2, 3, ...) se la: (i-1) + (k-
1) * K.
                else
                {
                    X = (i - 1) + (k - 1) * K;
                }
                break;  // Thoat ra khoi viec tinh toan vi da tim duoc ket
qua
            }
            // Neu chua tim thay M thi cap nhat tong so phan tu va tiep tuc xet den doan
ke tiep
            tongsophantu = tongsophantu + sophantu_nhom_i;
        }
        // Buoc 5. Ghi ket qua ra file
        ofstream fo("MODK.OUT");     // Mo file de ghi
        fo << X;                     // Ghi ket qua ra file
        fo.close();                  // Dong file sau khi ghi xong
        return 0;
}
```

5. Các bộ Test:

Mỗi câu của đề thi đều có đủ 10 bộ Test, với các giá trị đầu vào tương ứng với tỉ lệ (%) của các ràng buộc (Substack) mà đề bài yêu cầu. Những bộ Test này đã được gửi ở tệp tin đính kèm cùng với tài liệu này.

6. Kết quả chấm thử bằng phần mềm Themis 1.9, giới hạn thời gian chạy của mỗi bộ Test là 01 giây:

Tên bài	Điểm tối đa	Điểm chấm thử bằng Themis 1.9	
		Python	C++
Câu 1. Số giàu có (GIAUCO.*)	2.0 điểm	1.8 điểm (90%)	2.0 điểm (100%)
Câu 2. Dịch cúm (CORONA.*)	2.0 điểm	2.0 điểm (100%)	2.0 điểm (100%)
Câu 3. Cắt dây (CATDAY.*)	3.0 điểm	3.0 điểm (100%)	3.0 điểm (100%)
Câu 4. Sắp xếp theo modul K (MODK.*)	3.0 điểm	3.0 điểm (100%)	3.0 điểm (100%)
Tổng số điểm đạt được:		**9.8 điểm (98%)**	**10.0 điểm (100%)**

*- Bộ tài liệu này gồm hơn **700 trang**, được biên soạn nhằm phục vụ cho ôn luyện thi học sinh giỏi. Hiện tại, tác giả **xin chia sẻ trước 500 trang** để nhận ý kiến đóng góp và đánh giá từ cộng đồng.*

*- Nếu tài liệu nhận được sự quan tâm và đón nhận, **phần nội dung còn lại sẽ được tiếp tục chia sẻ miễn phí trong thời gian tới.***

- Rất mong nhận được ý kiến đóng góp chân thành của quý thầy cô và các bạn.