Lab 2: Mã hóa cổ điển

Nguyễn Văn Nhân

2025-04-07

1. Giới thiệu về bài thực hành

Trong bài thực hành này, sinh viên sử dụng C++ để thực hiện:

- Mã hóa đối xứng (Caesar Cipher).
- Mã hóa hoán vị (Rail Fence Cipher).
- **Lưu ý:** Trong các bước mã hóa ban đầu, dấu cách trong thông điệp sẽ bị bỏ trước khi mã hóa.

2. Mã hóa đối xứng – Caesar Cipher

2.1 Nhắc lại lý thuyết

- Mã Caesar dịch chuyển mỗi ký tự trong bảng chữ cái(tiếng Anh) một số vị trí cố định (khóa).

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**	**N**	**O**	**P**	**Q**	**R**	**S**	**T**	**U**	**V**	**W**	**X**	**Y**	**Z**

- Công thức mã hóa: $C = (P + k) \bmod 26$, giải mã: $P = (C - k) \bmod 26$.
- P: ký tự gốc(bản rõ), C: ký tự mã hóa(bản mã), k: khóa.
- Ví dụ:
    - Đầu vào(Bản rõ): "I LOVE YOU" → "ILOVEYOU" (bỏ dấu cách, in hoa).
    - Khóa: 3.
    - Mã hóa: "ILOVEYOU" → ta thu được bản mã: "LORYHBRX".

        I (8) → (8 + 3) mod 26 = 11 → L

        L (11) → (11 + 3) mod 26 = 14 → O

        O (14) → (14 + 3) mod 26 = 17 → R

        V (21) → (21 + 3) mod 26 = 24 → Y

        E (4) → (4 + 3) mod 26 = 7 → H

        Y (24) → (24 + 3) mod 26 = 1 → B

O (14) → (14 + 3) mod 26 = 17 → R

U (20) → (20 + 3) mod 26 = 23 → X

2.2 Thực hành

Bước 1: Nhập dữ liệu và chuẩn bị

Nhập thông điệp gốc và khóa, bỏ dấu cách trong thông điệp trước khi xử lý.

```
#include <iostream>
#include <string>
using namespace std;

// Hàm nhập dữ liệu từ người dùng
void input_data(string& plaintext, int& shift) {
    cout << "Nhap thong diep goc: ";
    getline(cin, plaintext);
    cout << "Nhap so vi tri dich chuyen (khoa): ";
    cin >> shift;
}

// Hàm loại bỏ dấu cách
string remove_spaces(string text) {
    string result = "";
    for (char c : text) {
        if (c != ' ') result += c;
    }
    return result;
}

// Hàm hiển thị thông tin đã chuẩn bị
void display_prepared_data(string plaintext, int shift) {
    cout << "Thong diep goc (sau khi bo dau cach): "<<plaintext << endl;
    cout << "Khoa: " << shift << endl;
}

int main() {
    string plaintext;
    int shift;

    // Bước 1: Nhập và chuẩn bị dữ liệu
    input_data(plaintext, shift);           // Nhập dữ liệu
    plaintext = remove_spaces(plaintext);   // Loại bỏ dấu cách
    display_prepared_data(plaintext, shift);// Hiển thị kết quả chuẩn bị

    return 0;
}
```

- **Chạy thử:** Nhập "I LOVE YOU" và 3. Quan sát đầu ra.

```
Nhap thong diep goc: I LOVE YOU
Nhap so vi tri dich chuyen (khoa): 3
Thong diep goc (sau khi bo dau cach): ILOVEYOU
Khoa: 3
```

Bước 2: Viết hàm mã hóa Caesar

- Mục tiêu của mã hóa Caesar

- Dịch chuyển từng ký tự trong chuỗi chữ cái theo một số lượng vị trí xác định (shift).
- Ví dụ:
    - Chuỗi gốc(Bản rõ): “HELLO”, shift = 3 → Bản mã: “KHOOR”
    - H: +3 → K
    - E: +3 → H
    - ...

- Cách mã hóa một ký tự

- Làm việc với chữ cái in hoa (có thể dùng toupper() để chuẩn hóa).
- Sử dụng bảng chữ cái tiếng Anh (A–Z có mã ASCII từ 65 đến 90).
- Áp dụng quy tắc:

$$C' = \left((C - \text{'A'} + \text{shift}) \mod 26\right) + \text{'A'}$$

Trong đó: C là ký tự gốc; 'A' là ký tự bắt đầu bảng chữ cái; $\mod 26$ đảm bảo quay vòng khi vượt qua Z.

- $C - \text{'A'}$:
  Chuyển ký tự thành chỉ số (0–25) bằng cách trừ mã ASCII của ký tự 'A'. Ví dụ, với 'H' (72), ta có: 72 - 65 = 7.
- $+\text{shift}$:
  Dịch chuyển ký tự bằng cách cộng số shift vào chỉ số vừa tính. Ví dụ, với 'H' và shift = 3, ta có: 7 + 3 = 10.
- $\mod 26$:
  Đảm bảo kết quả không vượt quá 'Z' (từ 0 đến 25). Nếu vượt quá, ta dùng phép chia lấy dư với 26. Ví dụ, nếu kết quả là 30, ta tính 30 % 26 = 4.

- +'A' :
  Chuyển chỉ số thành ký tự bằng cách cộng mã ASCII của 'A'. Ví dụ, với chỉ số 10, ta có: 10 + 65 = 75, và ký tự là 'K'.

- Vì sao cần bỏ dấu cách?

- Caesar chỉ xử lý ký tự chữ cái (a đến z, hoặc A đến Z).
- Dấu cách không mã hóa được nên sẽ bỏ trước khi mã hóa (trong bài toán đơn giản).

Thêm hàm mã hóa, xử lý trên thông điệp(bản rõ) đã bỏ dấu cách.

```
#include <iostream>
#include <string>
using namespace std;

// Hàm nhập dữ liệu từ người dùng
void input_data(string& plaintext, int& shift) {
    cout << "Nhap thong diep goc: ";
    getline(cin, plaintext);
    cout << "Nhap so vi tri dich chuyen (khoa): ";
    cin >> shift;
}

// Hàm loại bỏ dấu cách
string remove_spaces(string text) {
    string result = "";
    for (char c : text) {
        if (c != ' ') result += c;
    }
    return result;
}

// Hàm hiển thị thông tin đã chuẩn bị
void display_prepared_data(string plaintext, int shift) {
    cout << "Thong diep goc (sau khi bo dau cach): " << plaintext << endl;
    cout << "Khoa: " << shift << endl;
}

// Hàm mã hóa Caesar (chuẩn bị cho bước 2)
string caesar_encrypt(string plaintext, int shift) {
    string ciphertext = "";
    shift = shift % 26; // Đảm bảo rằng shift không vượt quá 26
    for (char c : plaintext) {
```

```
        if (isalpha(c)) { // Chỉ xử lý chữ cái
            c = toupper(c); // Chuyển ký tự về chữ hoa
            int ascii_code = (c - 'A' + shift) % 26 + 'A';
            ciphertext += (char)ascii_code;
        }
    }
    return ciphertext;
}

// Hàm giải mã
string caesar_decrypt(){
}


int main() {
    string plaintext;
    int shift;

    // Bước 1: Nhập và chuẩn bị dữ liệu
    input_data(plaintext, shift);
    plaintext = remove_spaces(plaintext);
    display_prepared_data(plaintext, shift);

    // Bước 2:Gọi hàm mã hóa
    string ciphertext = caesar_encrypt(plaintext, shift);
    cout << "Thong diep ma hoa: " << ciphertext << endl;

    return 0;
}
```

- **Chạy thử:** Nhập "I LOVE YOU" và 3. Kết quả: "LORYHBXR".

```
Nhap thong diep goc: I LOVE YOU
Nhap so vi tri dich chuyen (khoa): 3
Thong diep goc (sau khi bo dau cach): ILOVEYOU
Khoa: 3
Thong diep ma hoa: LORYHBRX
```

Yêu cầu

- **Q1:** Phân tích cách hàm caesar_encrypt xử lý ký tự và chạy thử với "HELLO WORLD", khóa 5.
- **Q2:** Thực hiện tinh chỉnh code để xử lý cả ký tự chữ thường không phải in hoa.
- **Q3:** Thực hiện hoàn thiện chương trình với việc bổ sung hàm giải mã.

- **Q4:** Điều chỉnh chương trình Caesar để xử lý dấu cách trong thông điệp gốc. Thay vì bỏ dấu cách trước khi mã hóa, giữ nguyên dấu cách và mã hóa cả chúng.

  Gợi ý: Giữ dấu cách trong plaintext và thêm vào ciphertext không thay đổi.

3. Mã hóa hoán vị – Rail Fence Cipher

3.1 Nhắc lại lý thuyết

- Rail Fence Cipher sắp xếp ký tự theo dạng zigzag trên các "đường ray", sau đó đọc theo từng ray để mã hóa.
- **Ví dụ:** Với thông điệp "I LOVE YOU" (bỏ dấu cách thành "ILOVEYOU") và 2 đường ray, quá trình zigzag diễn ra như sau:

```
Ray 0: I       O       E       O
         \     / \     / \     / \
Ray 1:     L       V     Y       U
```

  - Ký tự được đặt lần lượt theo dạng zigzag: I (ray 0) → L (ray 1) → O (ray 0) → V (ray 1), v.v.
  - Đọc theo từng ray: Ray 0: "IOEO", Ray 1: "LVYU".
  - Kết quả mã hóa: "IOEOLVYU".

3.2 Thực hành

Bước 1: Nhập dữ liệu và chuẩn bị

Nhập thông điệp(bản rõ) và số đường ray, bỏ dấu cách trong thông điệp.

```
#include <iostream>
#include <string>
using namespace std;

// Hàm nhập dữ liệu từ người dùng
void input_data(string& plaintext, int& rails) {
    cout << "Nhap thong diep goc: ";
    getline(cin, plaintext);
    cout << "Nhap so duong ray: ";
    cin >> rails;
}

// Hàm loại bỏ dấu cách
string remove_spaces(string text) {
```

```
    string result = "";
    for (char c : text) {
        if (c != ' ') result += c;
    }
    return result;
}

// Hàm hiển thị thông tin đã chuẩn bị
void display_prepared_data(string plaintext, int rails) {
    cout << "Thong diep goc (sau khi bo dau cach): " << plaintext << endl;
    cout << "So duong ray: " << rails << endl;
}

int main() {
    string plaintext;
    int rails;

    // Bước 1: Nhập và chuẩn bị dữ liệu
    input_data(plaintext, rails);           // Nhập dữ liệu
    plaintext = remove_spaces(plaintext);   // Loại bỏ dấu cách
    display_prepared_data(plaintext, rails);// Hiển thị kết quả chuẩn bị

    return 0;
}
```

- **Chạy thử:** Nhập "I LOVE YOU" và 2.

```
Nhap thong diep goc: I LOVE YOU
Nhap so duong ray: 2
Thong diep goc (sau khi bo dau cach): ILOVEYOU
So duong ray: 2
```

Bước 2: Viết hàm mã hóa Rail Fence

- Cách xây dựng thuật toán Zigzag

- Dùng một vector<string> fence với số phần tử = số đường ray.
- Duyệt từng ký tự:
    - Đặt ký tự vào đúng "đường ray".
    - Điều khiển hướng đi:
        - Nếu đang đi xuống và chạm đáy → đổi chiều lên trên.
        - Nếu đang đi lên và chạm đỉnh → đổi chiều xuống dưới.

    - Cần biến rail (vị trí hiện tại) và direction (hướng đi: +1 là xuống, -1 là lên).
- Cuối cùng nối các dòng lại để tạo thành chuỗi mã hóa.

- Xử lý tiền xử lý (bỏ dấu cách)

- Vì thuật toán không xử lý ký tự đặc biệt, nên ta cần:
    - Xóa các dấu cách để đơn giản hóa việc mã hóa.
    - Sau này nếu nâng cấp, có thể giữ lại vị trí các ký tự đặc biệt.

Thêm hàm mã hóa với logic zigzag 2 đường ray, xử lý trên thông điệp đã bỏ dấu cách.

```
#include <iostream>
#include <string>
#include <vector>
using namespace std;

// Hàm nhập dữ liệu từ người dùng
void input_data(string& plaintext, int& rails) {
    cout << "Nhap thong diep goc: ";
    getline(cin, plaintext);
    cout << "Nhap so duong ray: ";
    cin >> rails;
}

// Hàm loại bỏ dấu cách
string remove_spaces(string text) {
    string result = "";
    for (char c : text) {
        if (c != ' ') result += c;
    }
    return result;
}

// Hàm hiển thị thông tin đã chuẩn bị và kết quả mã hóa
void display_result(string plaintext, int rails, string ciphertext) {
    cout << "Thong diep goc (sau khi bo dau cach): " << plaintext << endl;
    cout << "So duong ray: " << rails << endl;
    cout << "Thong diep ma hoa: " << ciphertext << endl;
}

// Hàm mã hóa Rail Fence
string rail_fence_encrypt(string plaintext, int rails) {
```

```
    vector<string> fence(rails, ""); // Tạo vector chứa các đường ray
    int rail = 0;                    // Vị trí đường ray hiện tại
    int direction = 1;               // Hướng đi: 1 (xuống), -1 (lên)

    // Đặt từng ký tự vào đường ray theo pattern zigzag
    for (char c : plaintext) {
        fence[rail] += c;
        rail += direction;
        if (rail == rails - 1 || rail == 0) direction = -direction;
    }

    // Ghép các đường ray thành chuỗi mã hóa
    string ciphertext = "";
    for (int i = 0; i < rails; i++) {
        ciphertext += fence[i];
    }
    return ciphertext;
}


// Hàm giải mã Rail Fence
string rail_fence_encrypt(){

}

int main() {
    string plaintext;
    int rails;

    // Bước 1: Nhập và chuẩn bị dữ liệu
    input_data(plaintext, rails);
    plaintext = remove_spaces(plaintext);

    // Bước 2: Mã hóa
    string ciphertext = rail_fence_encrypt(plaintext, rails);

    // Hiển thị kết quả
    display_result(plaintext, rails, ciphertext);

    return 0;
}
```

- **Chạy thử:** Nhập "I LOVE YOU" và 2. Kết quả: "IOEOLVYU".

```
Nhap thong diep goc: I LOVE YOU
Nhap so duong ray: 2
Thong diep goc (sau khi bo dau cach): ILOVEYOU
```

```
So duong ray: 2
Thong diep ma hoa: IOEOLVYU
```

Yêu cầu

- **Q1:** Phân tích cách hàm rail_fence_encrypt tạo zigzag, thử thay đổi số đường ray thành 4 và quan sát kết quả.
- **Q2:** Chạy thử với từ khác "HELLO FIT DNU", 3 đường ray.
- **Q3:** Hoàn thiện chương trình với hàm giải mã.
- **Q4:** Điều chỉnh chương trình Rail Fence để xử lý dấu cách trong thông điệp gốc. Thay vì bỏ dấu cách trước khi mã hóa, giữ nguyên dấu cách và mã hóa cả chúng.

  Gợi ý: Với Rail Fence Cipher: Khi sắp xếp zigzag, giữ dấu cách như một ký tự bình thường, không bỏ đi.

4. Bài tập phát triển thêm

- **Q5:** Kết hợp mã Caesar (khóa 3) và Rail Fence (2 đường ray) cho "I LOVE YOU".
- **Q6:** Thêm kiểm tra đầu vào: chỉ chấp nhận chữ cái và dấu cách.
- **Q7:** Điều chỉnh chương trình để đọc thông điệp từ file input.txt thay vì nhập từ bàn phím.