A map of the world

Description automatically generated with low confidence

Sinh viên thực hiện:

* Lê Mậu Anh Phong

MSSV: 21520087

Lớp: ATTN2021

Người hướng dẫn:

* Nguyễn Thanh Sơn
* Nguyễn Đức Vũ

**ĐỒ ÁN CUỐI KÌ**

**MÔN CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT IT003**

**ĐỀ TÀI: PHÁ MÃ MẬT KHẨU ĐƯỢC MÃ HÓA BẰNG SUBSET SUM**

Thành phố Hồ Chí Minh – Năm 2022

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

------------------------------------------

**I. Giới thiệu:**

Mật khẩu là một tài nguyên quan trọng của mỗi cá nhân hay tổ chức. Nó được sử dụng để truy cập vào các tài nguyên quan trọng đối với từng mục tiêu cụ thể. Vì vậy việc bảo vệ mật khẩu là vô cùng cần thiết. Về phía hệ thống, khi nhận được tên đăng nhập và mật khẩu thì hệ thống cần phải lưu vào đâu đó có thể là cơ sở dữ liệu. Tuy nhiên mật khẩu trước khi được lưu cần phải được mã hóa nhằm tránh một người nào đó có quyền truy cập vào cơ sở dữ liệu có thể đọc được. Khi người dùng gửi mật khẩu lên để đăng nhập, hệ thống sẽ mã hóa rồi kiểm tra xem có giống với kết quả lúc đăng kí không. Nếu trùng khớp, người dùng sẽ được phép truy cập vào hệ thống.

Bước mã hóa thường có một số tính chất sau:

* Hàm mã hóa có thể thực hiện trong thời gian đa thức.
* “Không thể” đảo ngược được quá trình mã hóa. “Không thể” ở đây có thể là rất khó, tốn rất nhiều tài nguyên và thời gian để đảo ngược.
* Hạn chế việc xảy ra đụng độ. Việc hai mật khẩu khác nhau đi qua hàm mã hóa cho ra kết quả giống nhau được gọi là đụng độ.

Một bài toán ứng dụng trong việc xây dựng các thuật toán phổ biến được nghiên cứu rất nhiều đó là bài toán subset-sum. Bài toán được phát biểu như sau: Cho số nguyên mỗi số có độ dài bit và số nguyên . Tìm tập } sao cho .

Sau đây ta sẽ tìm hiểu một số vấn đề trong bài toán đó.

**II. Nội dung:**

1. **Đặt vấn đề:**

|  |  |  |
| --- | --- | --- |
| **#** | **8-char key** | **40-bit key** |
| 0 | 4szcirzz | 1111010010110010001001000100011100111001 |
| 1 | ibsalkc2 | 0100000001100100000001011010100001011100 |
| 2 | usnfujjz | 1010010010011010010110100010010100111001 |
| 3 | bu4mmejr | 0000110100111100110001100001000100110001 |
| 4 | 5bd5qclw | 1111100001000111111110000000100101110110 |
| 5 | 5vkmuyjo | 1111110101010100110010100110000100101110 |
| 6 | v521cfzw | 1010111111111001101100010001011100110110 |
| 7 | ty12gjq2 | 1001111000110111110000110010011000011100 |
| 8 | 35b5hs0a | 1110111111000011111100111100101101000000 |
| 9 | 0q0w2nna | 1101010000110101011011100011010110100000 |
| 10 | uwnzdxew | 1010010110011011100100011101110010010110 |
| 11 | xmjjhv3s | 1011101100010010100100111101011110110010 |
| 12 | lb3ffann | 0101100001111010010100101000000110101101 |
| 13 | za5lkafk | 1100100000111110101101010000000010101010 |
| 14 | dniohm4x | 0001101101010000111000111011001111010111 |
| 15 | 20uqpma2 | 1110011010101001000001111011000000011100 |
| 16 | 4ac0ist0 | 1111000000000101101001000100101001111010 |
| 17 | 2c24x3h2 | 1110000010111001111010111111010011111100 |
| 18 | c4u2phiq | 0001011110101001110001111001110100010000 |
| 19 | zdvpqfb0 | 1100100011101010111110000001010000111010 |
| 20 | wiszzamb | 1011001000100101100111001000000110000001 |
| 21 | 0nbn5c0v | 1101001101000010110111111000101101010101 |
| 22 | aipxjpv4 | 0000001000011111011101001011111010111110 |
| 23 | iphcrucy | 0100001111001110001010001101000001011000 |
| 24 | 3leopdkz | 1110101011001000111001111000110101011001 |
| 25 | 3wzkljly | 1110110110110010101001011010010101111000 |
| 26 | robtlkyw | 1000101110000011001101011010101100010110 |
| 27 | h0bjkm2m | 0011111010000010100101010011001110001100 |
| 28 | nniflpbf | 0110101101010000010101011011110000100101 |
| 29 | wvo3no2d | 1011010101011101110101101011101110000011 |
| 30 | kg4ts5y2 | 0101000110111101001110010111111100011100 |
| 31 | b12qgbzu | 0000111011111001000000110000011100110100 |
| 32 | wwahbmfq | 1011010110000000011100001011000010110000 |
| 33 | fu15rfbv | 0010110100110111111110001001010000110101 |
| 34 | hv2as24l | 0011110101111000000010010111001111001011 |
| 35 | vw3dilsf | 1010110110111010001101000010111001000101 |
| 36 | 1dcqok04 | 1101100011000101000001110010101101011110 |
| 37 | owryddqr | 0111010110100011100000011000111000010001 |
| 38 | kvjwuzgn | 0101010101010011011010100110010011001101 |
| 39 | ieae2oph | 0100000100000000010011100011100111100111 |

**Bảng 1.** Ví dụ về bảng cho mật khẩu có độ dài là 8 sử dụng bảng chữ cái 32 ký tự.

Cho một bảng gồm số nguyên dương với chiều dài n bit. Hệ thống mã hóa luôn được cung cấp mật khẩu với chiều dài cụ thể bit. Để mã hóa mật khẩu, hệ thống mã hóa sử dụng mật khẩu để chọn các số nguyên trong bảng và tính tổng của chúng sau đó chia lấy dư cho sẽ được kết quả mã hóa. Chúng ta sẽ chọn dòng thứ trong bảng tương ứng nếu bit thứ trong mật khẩu được bật.

Trong đề tài này, chúng ta sẽ sử dụng bảng chữ cái gồm 32 ký tự (“abcdefghijklmnopqrstuvwxyz012345”) trong các mật khẩu và mã hóa chúng bằng số nguyên 5-bit với 0 mã hóa ‘a’, 1 mã hóa ‘b’, và tương tự cho các ký tự tiếp theo. Như vậy, theo quy ước ban đầu, trong đó là số ký tự trong mật khẩu.

Chúng ta sẽ làm quen bằng ví dụ sau. Chúng ta sẽ sử dụng bảng như đã trình bày trong Bảng 1. Đầu tiên, chúng ta sẽ mã hóa mật khẩu “aaaaaaaa”, lấy từng ký tự của nó đổi sang hệ nhị phân rồi nối lại với nhau ta có 0000000000000000000000000000000000000000 sẽ được dạng nhị phân của “aaaaaaaa”. Như vậy không có bit nào trong “aaaaaaaa” được bật nên “aaaaaaaa” được mã hóa thành “aaaaaaaa”. Chúng ta sẽ thử tiếp với “bbbbbbbb”, mật khẩu này được biểu diễn trong hệ nhị phân là 0001000010000100001000010000100001000010. Như vậy các bit thứ 4, 9, 14, 19, 24, 29, 34, 39 (tính từ trái sang) trong “bbbbbbbb” được bật. Cho nên chúng ta sẽ lấy dòng thứ 4, 9, 14, 19, 24, 29, 34, 39 trong bảng tính tổng và chia lấy dư cho như sau:

|  |  |  |
| --- | --- | --- |
| **#** | **8-char key** | **40-bit key** |
| 4 | 5bd5qclw | 1111100001000111111110000000100101110110 |
| 9 | 0q0w2nna | 1101010000110101011011100011010110100000 |
| 14 | dniohm4x | 0001101101010000111000111011001111010111 |
| 19 | zdvpqfb0 | 1100100011101010111110000001010000111010 |
| 24 | 3leopdkz | 1110101011001000111001111000110101011001 |
| 29 | wvo3no2d | 1011010101011101110101101011101110000011 |
| 34 | hv2as24l | 0011110101111000000010010111001111001011 |
| 39 | ieae2oph | 0100000100000000010011100011100111100111 |
|  | z3mfp5nv | 1001100111101011000010101111111110110110101 |

Như vậy, sử dụng bảng như đề cập trong Bảng 1, mật khẩu “bbbbbbbb” sẽ được mã hóa thành “z3mfp5nv”.

Vậy là tôi đã trình bày xong về cách mã hóa. Các yêu cầu đặt ra:

* Viết chương trình đầu nhận vào mật khẩu đã được mã hóa gồm ký tự ( có giá trị tối đa là 12) sử dụng bảng chữ cái 32 ký tự theo phương pháp mô tả ở trên. Ngoài ra, chương trình còn được cho biết trước bảng T gồm dòng, mỗi dòng là một -bit key nào đó. Phân tích độ phức tạp của chương trình theo . Đầu ra của chương trình là mật khẩu trước khi được mã hóa.
* Việc sinh bảng không tốt có thể dẫn tới tồn tại trường hợp hai mật khẩu khác nhau có cùng có cùng mật khẩu được mã hóa. Trình bày một hướng giải quyết để tạo bảng sao cho không còn tồn tại trường hợp đó nữa.

1. **Giải quyết yêu cầu đặt ra:**

a. Yêu cầu thứ nhất:

**Nhận xét:** Việc mã hóa thực chính là việc chọn chọn ra một tập con của của bảng rồi tính tổng . Còn việc giải mã thực chất chính là việc tìm tập con có tổng bằng một số cho trước. Việc phá mã được coi là hoàn tất khi tìm thấy một tập con thỏa mãn.

**Quy ước:**

* Đặt .
* Ta chỉ sử dụng dạng biểu diễn nhị phân nên bảng lúc này chỉ cần sử dụng đến cột thứ , nên sẽ dùng mảng để chỉ cột thứ của bảng .
* Mật khẩu cần giải mã được đặt là , dạng nhị phân của kí hiệu là . Với ví dụ ở trên thì và
* .

Bài toán phá mã có thể được phát biểu lại như sau: Cho mảng gồm số nguyên và số nguyên , tìm một tập con của sao cho tổng bằng ().

**Các bước chính cần làm là:**

|  |
| --- |
| Bước 1: Đọc vào input.  Bước 2: Tạo mảng từ bảng .  Bước 3: Tính từ .  Bước 4: Tìm tập con của mà tổng bằng , nếu có thì lưu lại tập con đó và chuyển sang bước 5, ngược lại thử tập con tiếp theo.  Bước 5: Từ kết quả lưu lại chuyển thành mật khẩu. |

Các bước 1, 2, 3, 5 tương đối đơn giản, nên sẽ không được trình bày. Phần viết dưới sẽ trình bày cách kiểm tra có tồn tại không, từ đó tạo cơ sở để có thể truy vết được kết quả.

Bài toán này với phiên bản không có đã được nghiên cứu từ rất lâu và có nhiều thuật toán để giải. Sau khi tham khảo thì tôi thấy có một vài thuật toán:

|  |  |  |
| --- | --- | --- |
| Thuật toán | Độ phức tạp thời gian | Độ phức tạp không gian |
| Horowitz and Sahni |  |  |
| Schroeppel and Shamir |  |  |

Schroeppel and Shamir là phù hợp nhất điều kiện của đề bài, dưới đây tôi xin trình bày hướng giải dựa trên Schroeppel and Shamir để giải bài toán có .

**Ý chính của giải thuật:**

Bước 1: Chia thành 4 phần như sau: đặt ,

Bước 2: Sử dụng đệ quy quay lui để tìm tổng và cách chọn của từng tập con () trong lưu vào mảng .

Làm tương tự lần lượt với , , tương ứng với , , .

*Khi tạo V0, V1, V2, V3 thì lưu theo dạng {sum, state} với sum là tổng và state là chuỗi bit ở hệ thập phân biểu thị cách chọn của tập hợp để truy vết. Với cách lưu này khi ta cần kết hợp , thì sẽ có là , tương tự đôi với các tập khác.*

Bước 3: Sắp xếp tăng dần theo .

Bước 4: Giải bài toán: tìm , , , mà ( tức là lấy phần của biến ). Giả sử tìm được thì ta có cách chọn thỏa mãn cả đề bài sẽ là .

Để giải quyết được vấn đề ở Bước 3, đầu tiên ta cần xây dựng một cấu trúc dữ liệu priority queue đặc biệt:

Cho hai tập , và

Gọi là các phần tử của theo thứ tự tăng dần. Ta dễ dàng thấy rằng lên tới . Cấu trúc dữ liệu cần xây dựng gọi là inc (đầu vào là hai tập , ) hỗ trợ truy vấn inc.next() mà với lần gọi thứ () sẽ cho ta giá trị của , với thì trả về EMPTY. Hơn nữa không gian lưu trữ của nó là . Mã giả cách xây dựng như sau:

|  |
| --- |
| **Preprocessing** inc **with two sets , as input:**   1. Sort , . 2. Initialize priority queue . 3. For every add to with priority .   **Operation** inc.next()**:**   1. **if** is empty **then return** EMPTY. 2. Let be the lowest priority in the queue . 3. **while** the lowest priority in is **do** 4. Let be the element with the lowest priority in queue and remove it. 5. **if** then add with priority to . 6. **return** |

Với mã giả trên, output cótheo thứ tự sẽ là một dãy tăng dần (Lưu ý là độ ưu tiên khi thêm vào hàng đợi ưu tiên không được mod vì nếu mod output sẽ không còn một dãy tăng dần). Bước khởi tạo tốn thời gian . Ta thấy rằng sau bước khởi tạo , mỗi lần xóa đi một phần tử trong thì ta lại có cơ hội thêm vào nhiều nhất một phần tử nên do đó độ phức tạo không gian của inclà. Ngoài ra, vòng lặp bên trong inc.next()lặp lần là sẽ rỗng, mỗi lần lặp thực hiện xóa và thêm vào (tốn cho các thao tác trên priority queue) nên tổng thời gian thực hiện của cấu trúc dữ liệu trên là ).

Với cách làm tương tự ta có thể tạo ra một cấu trúc dữ liệu với output là một dãy giảm dần gọi là dec**.**

Ta sẽ tạo inc với đầu vào là và để liệt kê các phần tử của (theo thứ tự tăng dần) và sử dụng dec để liệt kê các phần tử của (theo thứ tự giảm dần). Ta cần thay đổi một chút để thêm các vào.

Tiếp theo với mỗi và duyệt qua bằng kỹ thuật hai con trỏ, đặt , , tính và so sánh với . Nếu thì có luôn cách giải (cách chọn , không cần tìm nữa), nếu thì tiếp tục gán . Nếu inc hoặc dec rỗng mà vẫn chưa tìm được cách giải thì không có cách giải nào thỏa mãn.

Tuy nhiên vẫn chưa dừng lại ở đó, ta nhớ rằng cái ta thực sự cần là chứ không phải là (không có mod). Vì vậy ta cần biến đổi từ dạng có mod sang dạng không có mod:

(với số nguyên nào đó).

Vì , là tổng hai số nguyên không âm nhỏ hơn , nên tối đa . Nên tối đa . Nên ta có là số nguyên không âm nhỏ hơn . Do đó sau khi thực hiện thuật toán trên với , ta gán và thực hiện ba lần như vậy.

Dưới đây là cài đặt chi tiết để tìm một nghiệm của cả yêu cầu thứ nhất, nếu muốn ghi ra tất cả các nghiệm thì cần sửa lại một vài chỗ (nhường lại cho bạn đọc).

#include <bits/stdc++.h>

using namespace std;

const string ALPHABET = "abcdefghijklmnopqrstuvwxyz012345";

int n;

long long hash\_num = 0, s[100], MOD;

string hashed\_message;

vector <pair <long long, long long> > V[4];

bool getBit(long long v, int p)

{

    return (v >> p) & 1;

}

long long turnOn(long long v, int p)

{

    return (1LL << p) | v;

}

void input()

{

    cin >> n;

    n \*= 5;

    MOD = 1LL << n;

    long long sum = 0;

    for (int i = 0; i < n; ++i)

    {

        string x;

        cin >> x;

        string y = "";

        for (auto c: x)

        {

            int v = ALPHABET.find(c);

            for (int j = 4; j >= 0; --j)

                y += (char)(48 + getBit(v, j));

        }

        reverse(y.begin(), y.end());

        for (int j = 0; j < n; ++j)

            if (y[j] == '1')

                s[i] += 1LL << j;

    }

}

class inc

{

    private:

        vector <pair <long long, long long> > \*a, \*b;

        priority\_queue <pair <pair <long long, long long>, pair <int, int> > > pq;

    public:

        inc(vector <pair <long long, long long> > \*a, vector <pair <long long, long long> > \*b)

        {

            this->a = a;

            this->b = b;

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

                pq.push({{- ((\*a)[0].first + (\*b)[i].first), (\*a)[0].second + (\*b)[i].second}, {0, i}});

        }

        pair <long long, long long> top()

        {

            if (pq.empty()) return {LLONG\_MAX, LLONG\_MAX};

            long long w = pq.top().first.first;

            long long state = pq.top().first.second;

            while (pq.size() && pq.top().first.first == w)

            {

                auto p = pq.top(); pq.pop();

                int i = p.second.first, j = p.second.second;

                if (i < a->size() - 1)

                    pq.push({{- ((\*a)[i + 1].first + (\*b)[j].first), (\*a)[i + 1].second + (\*b)[j].second}, {i + 1, j}});

            }

            return {- w, state};

        }

};

void calc\_hash\_num()

{

    cin >> hashed\_message;

    reverse(hashed\_message.begin(), hashed\_message.end());

    long long pw32 = 1;

    for (int i = 0; i < n/5; ++i)

    {

        int k = ALPHABET.find(hashed\_message[i]), x = 0;

        hash\_num += k \* pw32;

        pw32 \*= 32LL;

    }

}

void backtrack(int k, int r, long long sum, long long state, int id)

{

    if (k > r)

        return V[id].push\_back({(id < 2 ? 1 : - 1) \* sum, state}), void();

    for (int i = 0; i < 2; ++i)

        if (i)

            backtrack(k + 1, r, (sum + s[k]) % MOD, turnOn(state, k), id);

        else

            backtrack(k + 1, r, sum, state, id);

}

void convert(long long stateX)

{

    string ans = "";

    for (int i = 0; i < n/5; ++i)

    {

        int order = 0;

        for (int j = 5 \* i, k = 4; j < 5 \* (i + 1); ++j, --k)

            order += getBit(stateX, j) \* (1 << k);

        ans += ALPHABET[order];

    }

    cout << ans;

    exit(0);

}

void two\_pointers(long long hash\_num)

{

    inc lPart(&V[0], &V[1]), rPart(&V[2], &V[3]);

    pair <long long, long long> last = {- 1, - 1};

    while (1)

    {

        auto l = lPart.top(), r = last;

        if (r.first == - 1) last = r = rPart.top();

        if (l.first == LLONG\_MAX || r.first == LLONG\_MAX)

            break;

        while (1)

        {

            if (r.first == LLONG\_MAX) return;

            if (hash\_num - l.first == - r.first) convert(l.second + r.second);

            else if (hash\_num - l.first > - r.first)

            {

                last = r;

                break;

            }

            last = r = rPart.top();

        }

    }

}

int main()

{

    freopen("input.txt", "r", stdin);

    input();

    calc\_hash\_num();

    backtrack(0, n / 4 - 1, 0, 0, 0);

    backtrack(n / 4, n / 2 - 1, 0, 0, 1);

    backtrack(n / 2, n / 2 + n / 4 - 1, 0, 0, 2);

    backtrack(n / 2 + n / 4, n - 1, 0, 0, 3);

    for (int i = 0; i < 4; ++i) sort(V[i].begin(), V[i].end());

    two\_pointers(hash\_num);

    two\_pointers(hash\_num + MOD);

    two\_pointers(hash\_num + 2 \* MOD);

    two\_pointers(hash\_num + 3 \* MOD);

    cout << - 1;

    return 0;

}

**Phân tích độ phức tạp:**

Phần input(): duyệt lần để tạo mảng , mỗi lần lại duyệt đọc vào một xâu ký tự, chuyển xâu đó sang nhị phân ( bit), rồi từ dạng nhị phân đó chuyển qua thập phân (bước này tốn ) lưu vào mảng nên độ phức tạp thời gian là và không gian là .

Phần calc\_hash\_num(): duyệt qua từng ký tự của mật khẩu cần phá mã nên độ phức tạp thời gian là , độ phức tạp không gian là vì chỉ lưu lại một vài biến thông thường.

Phần backtrack(): duyệt qua từng tập con của tập cha mà tập cha có số lượng phần tử không quá nên tốn thời gian . Mỗi tập con cần lưu lại thông tin thành mảng cho các bước sau nên tốn về không gian .

Phần sort(): dùng để sắp xếp lại mảng sau bước backtrack() nên về thời gian cần , chúng ta không dùng cấu trúc dữ liệu để lưu lại gì ở bước này nên về không gian là .

Phần two\_pointers(): tạo ra inc và dec cần thời gian và không gian (bước Preprocessing). Duyệt hai con trỏ (trình bày ở phần trước) qua từng phần tử của inc và dec mỗi phần tử một lần. Mà inc và dec được xây dựng là hai mảng có số lượng phần tử không quá nên inc và dec có số lượng phần tử tối đa khoảng nên cần . Tại bước này không lưu lại gì – độ phức tạp không gian .

Vì vậy ta có độ phức tạp của cả thuật toán về mặt thời gian là , không gian là .

**Thử nghiệm thực tế:** thực hiện trên máy 4 GB RAM. Để kiểm tra được tài nguyên cần dùng trong tình huống xấu nhất bộ test được sử dụng yêu cầu ghi ra hết nghiệm tìm được, tức là phải thử hết trong không gian tìm kiếm ([Test](https://khmt.uit.edu.vn/wecode/it003/assignment/101/856)). Source code có thể được tìm thấy trong link github ở phần cuối.

|  |  |  |  |
| --- | --- | --- | --- |
| **Thứ tự** | **Độ dài mật khẩu** | **Thời gian phá mã** | **Không gian cần dùng** |
| Test 1 |  | < 0.01 s | < 1 KiB |
| Test 2 |  | < 0.01 s | < 1 KiB |
| Test 3 |  | 0.69 s | 3060 KiB |
| Test 4 |  | 31.18 s | 6436 KiB |
| Test 5 |  | 1246.09 s | 6436 KiB |
| Test 6 |  | 0.75 s | 2128 KiB |

b. Yêu cầu thứ hai:

Dạng nhị phân của mật khẩu sau mã hóa chính là kết quả của việc tính tổng (mod ) của một tập con của mảng . Vì vậy yêu cầu của câu này tương đương với việc giải bài toán: Xây dựng một mảng gồm phần tử sao cho tổng mọi tập con của (mod ) đôi một khác nhau. Nên tôi sẽ trình bày cách giải bài toán này thay cho bài toán gốc.

Ý tưởng ban đầu là xây dựng mảng với . Lúc này với cách chọn được biểu diễn bằng dãy bit sẽ có tổng được biểu diễn dưới dạng nhị phân cũng chính là .

Ví dụ: , . Nếu ta chọn tập được biểu diễn là thì sẽ có tổng là

Nên sẽ có tập có tổng tương ứng với chính cách chọn. Đã thỏa mãn được đề bài nhưng trên thực tế việc sử dụng mảng có tính chất rất dễ bị bẻ khóa kỹ thuật tham lam ([chi tiết](https://www.geeksforgeeks.org/find-the-subsequence-with-given-sum-in-a-superincreasing-sequence/) sẽ không được trình bày ở đây). Nên cần tìm cách cải tiến, cách của tôi như sau:

Bằng thực nghiệm trên một vài ví dụ tôi thấy rằng mình có thể cải tiến bằng cách xây dựng một mảng với là một số nguyên không âm, rồi gán mod . Tránh chọn với mọi vì sẽ rơi vào cách tiếp cận ban đầu.

Lấy ví dụ với , với cách tiếp cận , ta xếp các tập con lên bảng:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Cách chọn |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Tổng |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Với cách tiếp cận thứ hai, bản chất chính là biến thành nên nếu ở cách tiếp cận ban đầu có tổng là mà lại có có trong đó (ví dụ thì có , , ở trong) thì sau phép biến đổi trở thành với mọi .

Ví dụ có:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Vì , , đều bằng nên không thay đổi gì, nên sẽ chỉ xét , những số được tô tím là số có trong biểu diễn nhị phân:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Cách chọn |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Trước |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | Những số màu tím là những số có trong cách biểu diễn nhị phân.  Toàn bộ số màu tím sau biến đổi được cộng thêm .  Còn số màu đen giữ nguyên. | | | | | | | | | | | | | | | |
| Sau | 0000 | 0011 | 0010 | 0101 | 0100 | 0111 | 0110 | 1001 | 1000 | 1011 | 1010 | 1101 | 1100 | 1111 | 1110 | 0001 |

Ta sẽ thử với ví dụ phức tạp hơn:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Với bảng này thì ta có những điều sau:

* Những số có trong biểu diễn nhị phân của nó sẽ được cộng thêm (mod ).
* Những số có trong biểu diễn nhị phân của nó sẽ được cộng thêm (mod MOD).
* Những số còn lại đứng yên.

Vậy với những số cả cả và trong biểu diễn nhị phân sẽ chịu ảnh hưởng từ cả hai điều đầu tiên (cộng ). Tóm lại:

* Những số có mà không có trong biểu diễn nhị phân của nó (màu tím) sẽ được cộng thêm lần (mod ).
* Những số có mà không có trong biểu diễn nhị phân của nó (màu xanh) sẽ được cộng thêm (mod ).
* Những số có cả và trong biểu diễn nhị phân của nó (màu vàng) sẽ được công thêm (mod ).
* Những số còn lại giữ nguyên.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Cách chọn |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Trước |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Sau |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Sau phép biến đổi đó không có 2 số nào nằm vào cùng một ô, tức là các tổng của tập con của đôi một khác nhau.

Tôi có suy đoán rằng: Việc cộng thêm đó làm cho các dải màu di chuyển sang trái theo vòng tròn theo một quy tắc nào đó. Sử dụng ví dụ 2 ta thấy:

* Dải màu tím được dịch trái 4 lần.
* Dải màu xanh được dịch trái 8 lần.
* Dải màu vàng được dịch sang trái 12 lần.
* Dải màu đen được dịch trái 0 lần.
* Các dải sau khi dịch không “đè” lên nhau.

Tuy nhiên lời giải thích đó vẫn chưa thuyết phục, sau đây là phần trình bày chứng minh bằng toán học:

|  |
| --- |
| Gọi và là hai vị trí bất kì trước khi biến đổi.  là tập hợp những vị trí chứa bit 1 trong dạng biểu diễn nhị phân của .  là tập hợp những vị trí chứa bit 1 trong dạng biểu diễn nhị phân của .  là sau biến đổi.  là sau biến đổi.  Ta có:        Ta cần chứng minh: nếu thì (hoặc )    ⬄  ⬄ (luôn tồn tại thỏa mãn)  ⬄ (1) |
| **Trường hợp 0: Nếu** , không cần chứng minh thêm. |
| **Trường hợp 1: Nếu ( hoặc ) và**  Không mất tính tổng quát, giả sử Ta có nên (1) trở thành    ⬄ (với là phần tử nhỏ nhất trong  Chia cả hai vế cho , ta được:    Do và nên vế trái sẽ là một số chẵn, còn vế phải sẽ là một số lẻ. Điều này trái với điều kiện của đề bài nên không thể xảy ra. |
| **Trường hợp 2: Các trường hợp còn lại**  Gọi là phần tử nhỏ nhất của .  là phần tử nhỏ nhất của .  Nếu , thì ta trừ hai vế của (1) cho rồi thay thế thành , thành và tiếp tục xét từ đầu.  Nếu , không mất tính tổng quát giả sử , ta có:  (1) ⬄  Sau đó, chia hai vế của (1) cho , ta có:    Vì , , nên vế trái là một số lẻ, vế phải là một số chẵn. Do đó không thể xảy ra trường hợp này.  Vì vậy . |
| Do đó hay .  Tức là với thì .  Vậy sau phép biển đổi ta có dãy số thỏa mãn đề bài. |

Tóm lại ta có một hướng xây dựng mảng mà tổng mọi tập con của nó đôi một phân biệt (không nói tới vấn đề bảo mật):

với là một số nguyên bất kì.

**III. Một số vấn đề khác:**

* Ta thấy rằng thời gian để phá mã tăng lên rất nhanh khi mật khẩu dài hơn đây chính là cơ sở của khuyến cáo nên đặt mật khẩu phức tạp và dài.

**IV. Tài liệu tham khảo:**

* [Subset sum problem](https://en.wikipedia.org/wiki/Subset_sum_problem).
* [Efficient Cryptographic Schemes Provably as Secure as Subset Sum](https://web.stevens.edu/algebraic/Files/SubsetSum/impagliazzo96efficient.pdf).
* [Improving Schroeppel and Shamir’s Algorithm for Subset Sum via Orthogonal Vectors](https://arxiv.org/pdf/2010.08576v2.pdf).
* [Improving Schroeppel and Shamir’s Algorithm for Subset Sum via Orthogonal Vectors (video)](https://www.youtube.com/watch?v=RUlIAc9Fqo4).

**V. Link github:** [link](https://github.com/lemauanhphong/Password-Cracking-SubsetSum)**.**