

ỦY BAN NHÂN DÂN THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC SÀI GÒN

KHOA CÔNG NGHỆ THÔNG TIN



BÁO CÁO BÀI TẬP NHÓM

MÔN: CƠ SỞ TRÍ TUỆ NHÂN TẠO

Nhóm môn học: 04

Mã lớp: DCT1194

Nhóm báo cáo: 04

3119560045 Phạm Đình Phương Nam

3119410181 Trần Văn Khang

3119410183 Phạm Tuấn Khanh

3119410204 Trần Hữu Khương

3119410215 Võ Hoàng Kiệt

3119410354 Lê Thái Thanh Sơn

TP. HỒ CHÍ MINH – THÁNG 4 NĂM 2022

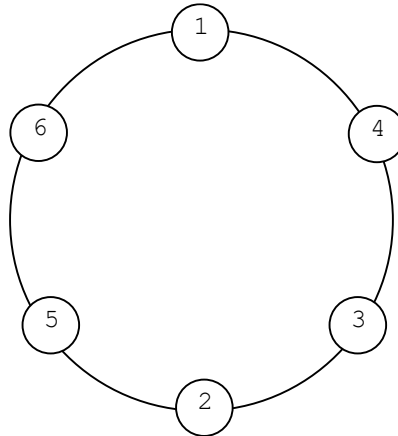
MỤC LỤC

006. VÒNG SỐ NGUYÊN TỐ	2
014. RẢI SỎI	13
029. SƠN CỘT.....	16
045. MÃ GRAY	19
054: THỨ TỰ TỪ ĐIỂN.....	21

Nội dung

006. VÒNG SỐ NGUYÊN TỐ

Một vòng tròn chứa $2n$ vòng tròn nhỏ (Xem hình vẽ). Các vòng tròn nhỏ được đánh số từ 1 đến n theo chiều kim đồng hồ. Cần điền các số tự nhiên từ 1 đến $2n$ mỗi số vào một vòng tròn nhỏ sao cho tổng của hai số trên hai vòng tròn nhỏ liên tiếp là số nguyên tố. Số điền ở vòng tròn nhỏ 1 luôn là số 1.



Dữ liệu: Vào từ file văn bản CIRCLE.INP chứa số nguyên dương n ($1 < n < 10$)

Kết quả: Ghi ra file văn bản CIRCLE.OUT:

- Dòng đầu tiên ghi số lượng các cách điền số tìm được (k).
- Dòng thứ i trong số k dòng tiếp theo ghi các số trong các vòng tròn nhỏ bắt đầu từ vòng tròn nhỏ 1 đọc theo thứ tự của các vòng tròn nhỏ

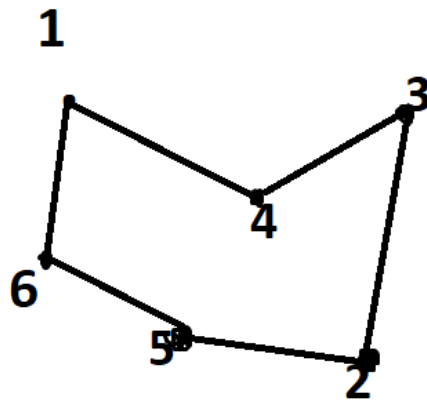
Ví dụ:

CIRCLE . INP	CIRCLE . OUT
3	2 1 4 3 2 5 6 1 6 5 2 3 4

CIRCLE . INP	CIRCLE . OUT
4	4 1 2 3 8 5 6 7 4 1 2 5 8 3 4 7 6 1 4 7 6 5 8 3 2 1 6 7 4 3 8 5 2

- **Ý tưởng** : Từ hình vẽ minh họa ta nhận thấy vòng tròn 1 có thể đi đến các vòng tròn có giá trị 4,6 Tương tự vòng tròn có giá trị 4 chỉ có thể đi đến vòng tròn giá trị 1 hoặc 3,.. Ta có thể xem như là đường đi giữa các đỉnh đồ thị.

Từ phân tích trên ta được đồ thị:



- Ví dụ $n = 3$, với $2n$ vòng tròn nhỏ (đề bài) thì ta được 6 đỉnh và bắt đầu từ đỉnh 1. Dễ dàng nhận thấy đồ thị này là đồ thị có hướng nên ta sẽ có được 2 chu trình :

Chu trình 1 : 1 4 3 2 5 6 1

Chu trình 2 : 1 6 5 2 3 4 1

- Tổng của 2 đỉnh kề nhau sẽ là 1 số nguyên tố :
+ Ví dụ đỉnh 1 và đỉnh 4 thì tổng của chúng bằng 5 là 1 số nguyên tố.

Dựa vào ý tưởng này ta xây dựng một ma trận kề có $2n * 2n$ phần tử. Bài này ta sẽ lấy ví dụ $n = 3$

Bước 1 : Ta xây dựng một hàm kiểm tra số nguyên tố.

Code :

```

1. bool isPrime(int n) // Hàm kiểm tra các số nguyên tố.
2. {
3.
4.     if (n <= 1)
5.         return false;
6.     for (int i = 2; i < sqrt(n) ; i++)
7.         if (n % i == 0)
8.             return false;
9.
10.        return true;
11.    }

```

Bước 2 : Khởi tạo ma trận $2n * 2n$ với các giá trị mặc định là 0 .

Code :

```

int A[100][100];

void init(int n) // khởi tạo giá trị mặc định của chu trình
{
    for (int i = 1; i <= 2 * n; i++)
    {
        for (int j = 1; j <= 2 * n; j++)
        {
            A[i][j] = 0;
        }
    }
}

```

Kết quả sẽ được một ma trận sau đây.

0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

$j = 1$ $i = 1$	1	2	3	4	5	6
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0
6	0	0	0	0	0	0

Bước 3 : Xây dựng ma trận kề với điều kiện $i + j$ sẽ là số nguyên tố.

Code :

```
void matran(int n) // khởi tạo ma trận kề
{
    for (int i = 1; i <= 2 * n; i++)
    {
        for (int j = 1; j <= 2 * n; j++)
        {
            int sum = i + j;
            if (isPrime(sum))
            { cout<<"i = "<<i <<"j = "<<j<<endl;
              A[i][j] = 1;
            }
        }
    }
}
```

- Ví dụ $i = 1, j = 4$ thì $(i + j) = 5$ sẽ là 1 số nguyên tố thì sẽ thay đổi giá trị của ma trận tại vị trí (i,j) từ 0 thành 1;
- Tương tự ta có :

$$i = 1, j = 4$$

$$i = 1, j = 6$$

$$i = 2, j = 5$$

$$i = 4, j = 1$$

$$i = 5, j = 2$$

$$i = 6, j = 1$$

Ma trận sau khi đã được thay đổi giá trị :

0	0	0	1	0	1
0	0	0	0	1	0
0	0	0	0	0	0
1	0	0	0	0	0
0	1	0	0	0	0
1	0	0	0	0	0

Bước 4 : Từ ma trận kề với $2n$ đỉnh duyệt từ đỉnh 1 ta áp dụng chu trình Hamilton :

Mã giả :

```

void Hamilton( int k) {

    /* Liệt kê các chu trình Hamilton của đồ thị bằng cách phát
    triển dãy đỉnh

    (X[1], X[2], ..., X[k-1] ) của đồ thị  $G = (V, E)$  */

    for y ∈ Ke (X[k-1]) {

        if (k==n+1) and (y == v0) then

            Ghinhan(X[1], X[2], ..., X[n], v0);

        else {

            X[k]=y; chuaxet[y] = false;

            Hamilton(k+1);

            chuaxet[y] = true;

        }

    }

}

```

Code :


```

#define MAX 100

int C[MAX], B[MAX];

int A[100][100];

int d; // số lượng chu trình

int sodinh ; // số đỉnh = 2n;

void Result(void) // in chu trình
{
    d++; // tăng biến đếm các chu trình có thể tìm được

    for (int i = sodinh; i > 0; i--)

        cout << B[i] << " ";

    cout << endl;
}

void Hamilton(int *B, int *C, int i) // duyệt chu trình Hamilton với
tham số đầu vào là Ma trận kề A
{
    int j, k;

    for (j = 1; j <= sodinh; j++)
    {
        if (A[B[i - 1]][j] == 1 && C[j] == 0)
        {
            B[i] = j;
            C[j] = 1;

            if (i < sodinh)
                Hamilton(B, C, i + 1);

            else if (B[i] == B[0])
                Result();

            C[j] = 0;
        }
    }
}

```

```

    }
}

```

=> Kết quả trả về sẽ là các chu trình xuất phát từ đỉnh có thể đi được trong đồ thị :

```

1 6 5 2 3 4
1 4 3 2 5 6
Press any key to continue . . .

```

- Tương tự với $1 < n < 10$ ta sẽ được số lượng cách điền vào vòng tròn là :

N =	2	3	4	6	7	8	9
Số cách =	2	2	4	96	1024	2880	81024

Sau đây là mã nguồn của chương trình này :

- Cần tạo 2 file **CIRCLE.INP** và **CIRCLE.OUT**
- Code :

```

1. #include <iostream>
2. #include <cstdio>
3. #include <cstdlib>
4. #include <algorithm>
5. #include <cmath>
6. #include <string>
7. #include <stdlib.h>
8. #include <fstream>
9. #include <sstream>
10.
11.     #define MAX 100
12.     using namespace std;
13.
14.     int A[MAX][MAX];

```

```

15.     int C[MAX], B[MAX];
16.     int n;
17.     int d;          // số lượng chu trình
18.     int sodinh; // số đỉnh = 2n;
19.     string s;
20.
21.     string numberToString(unsigned int n)
22.     {
23.         stringstream ss;
24.         ss << n;
25.         return ss.str();
26.     }
27.     void readFile() // hàm đọc file
28.     {
29.         ifstream file;
30.         file.open("CIRCLE.INP");
31.         file >> n;
32.         file.close();
33.     }
34.
35.     void writeFile()
36.     {
37.         // dữ liệu về các chu trình đã được gán vào biến s ở
        trong hàm Result();
38.         ofstream file("CIRCLE.OUT");
39.         file << d; // in số lượng chu trình tìm thấy
40.         file << '\n';
41.         file << s; // in biến s
42.         file.close();
43.     }
44.
45.     void init(int n) // khởi tạo giá trị mặc định của chu
        trình
46.     {
47.         for (int i = 1; i <= 2 * n; i++)
48.         {
49.             for (int j = 1; j <= 2 * n; j++)
50.             {
51.                 A[i][j] = 0;
52.             }
53.         }
54.     }
55.
56.     bool isPrime(int n) // Hàm kiểm tra các số nguyên tố.
57.     {
58.         if (n <= 1)
59.             return false;
60.         for (int i = 2; i <= sqrt(n); i++)
61.             if (n % i == 0)
62.                 return false;

```

```

63.
64.         return true;
65.     }
66.
67.     void matran(int n) // khởi tạo ma trận kề
68.     {
69.         for (int i = 1; i <= 2 * n; i++)
70.         {
71.             for (int j = 1; j <= 2 * n; j++)
72.             {
73.                 int sum = i + j;
74.                 int diff = abs(i - j);
75.                 if (isPrime(sum))
76.                 {
77.                     A[i][j] = 1;
78.                 }
79.             }
80.         }
81.     }
82.
83.     void Result(void) // in chu trình
84.     {
85.         d++;
86.         for (int i = sodinh; i > 0; i--)
87.         {
88.             cout << B[i] << " ";
89.             string geek = numberToString(B[i]); // chuyen so
                thanh chu roi
90.             s.append(geek); // gan vao
                bien "s" de luu vao file
91.             s.append(" ");
92.         }
93.         s.append("\n");
94.         // tăng biến đếm các chu trình có thể tìm được
95.         cout << endl;
96.     }
97.
98.     void Hamilton(int *B, int *C, int i) // duyệt chu trình
                Hamilton với tham số đầu vào là Ma trận kề A
99.     {
100.         int j, k;
101.         for (j = 1; j <= sodinh; j++)
102.         {
103.             if (A[B[i - 1]][j] == 1 && C[j] == 0)
104.             {
105.                 B[i] = j;
106.                 C[j] = 1;
107.                 if (i < sodinh)
108.                     Hamilton(B, C, i + 1);
109.                 else if (B[i] == B[0])

```

```

110.             Result();
111.             C[j] = 0;
112.         }
113.     }
114. }
115.
116. int main()
117. {
118.     B[0] = 1;
119.     readFile();
120.     int i = 1; // xuất phát từ đỉnh 1
121.     d = 0;     // gán biến đếm số lượng chu trình = 0 ;
122.     init(n);   // khởi tạo ma trận
123.     matran(n); // tạo ma trận kề
124.     sodinh = 2 * n;
125.     Hamilton(B, C, i); // chuyển ma trận kề thành chu
        trình
126.     writeFile();
127.     system("pause");
128.     return 0 ;
129. }

```

014. RẢI SỎI

Xét trò chơi rải sỏi với một người chơi như sau: Cho cây T và một đồng sỏi gồm K viên

ở mỗi bước người ta lấy 1 viên sỏi từ đồng sỏi và đặt vào một nút lá tùy chọn

Nếu nút p có r nút lá và tất cả và tất cả các nút lá đều có sỏi thì người ta gom tất cả các viên sỏi ở lá lại, đặt 1 viên ở nút p , xoá các nút lá của nó và hoàn trả $r - 1$ viên sỏi còn lại vào đồng sỏi.

Trò chơi kết thúc khi đã đặt được 1 viên sỏi vào nút gốc

Nhiệm vụ đặt ra là theo cấu trúc của cây T , xác định số viên sỏi tối thiểu ban đầu để trò chơi có thể kết thúc bình thường. Cây có n nút ($N \leq 400$), nút gốc được đánh số là 1.

Dữ liệu: vào từ file văn bản **STONE.INP**

- Dòng đầu: số n
- Dòng thứ i trong số n dòng tiếp theo có dạng: $i \ m \ i_1 \ i_2 \ \dots \ i_m$. Trong đó m là số nút con của nút i ; i_1, i_2, \dots, i_m : Các nút con của nút i .

Kết quả: đưa ra file **STONE.OUT** số lượng viên sỏi tối thiểu cần thiết

Ví dụ

STONE.IN	STONE.OU
P	T
7	3
1 2 2 3	
2 2 5 4	
3 2 6 7	

THUẬT TOÁN:

Cho dễ tưởng tượng đây là kiểu giống bài ô ăn quan.

Đầu tiên xác định được đây là bài toán Quy Hoạch Động trên cây

Cho $f[i]$ là số sỏi ít nhất cần có để đi hết các gốc i , tiếp theo đó ta sử dụng tìm kiếm theo chiều sâu đi từ trên xuống dưới (đi từ gốc tới các nút con).

Cách tính nút cha từ các nút con: ta sẽ lần lượt lấy các nút cần lớn rồi đi theo thứ tự giảm dần.

Vì dễ đi theo thứ tự giảm dần nên ta cho các nút con vào một mảng rồi sort nó theo thứ tự mà ta mong muốn, vì giới hạn nhỏ nên yên tâm.

Kết quả sẽ ra là f[1]

CODE:

```
#include <bits/stdc++.h>

using namespace std;

#define pb push_back

const int N = 500;

int n, f[N], b[N];

vector<int> a[N];

bool cmp(int u, int v) { return u > v; }

void dfs(int u)
{
    int m = a[u].size();
    if (m == 0) {
        f[u] = 1;
        return;
    }
    for (int i = 0; i < m; i++)
        dfs(a[u][i]);
    for (int i = 0; i < m; i++)
        b[i] = f[a[u][i]];
    sort(b, b + m, cmp);
    int ans = 0, now = 0;
    for (int i = 0; i < m; i++) {
        if (now < b[i]) {
            ans += b[i] - now;
```

```

        now = b[i] - 1;

    }

    else {

        now--;

    }

}

f[u] = ans;

}

int main()

{

    scanf("%d", &n);

    int m, u, v;

    while (scanf("%d", &u) == 1) {

        scanf("%d", &m);

        for (int i = 1; i <= m; i++)

            scanf("%d", &v), a[u].pb(v);

    }

    dfs(1);

    printf("%d", f[1]);

    return 0;}

```


029. SƠN CỘT

Trên một nền phẳng đã được chia thành các lưới ô vuông đơn vị gồm $m \times n$ ô ($m, n \leq 100$), người ta đặt chồng khít lên nhau các khối lập phương đơn vị thành những cột. Khối dưới cùng của cột chiếm trọn một ô của lưới. Chiều cao của mỗi cột được tính bằng số khối lập phương đơn vị tạo thành cột đó. Sau khi xếp xong toàn bộ các cột, người ta tiến hành sơn các mặt nhìn thấy được của các cột.

Yêu cầu: Biết chiều cao của mỗi cột, hãy tính số đơn vị diện tích cần sơn.

e

Dữ liệu vào đặt trong file văn bản PAINT.INP. Trong đó:

Dòng đầu tiên ghi hai số nguyên dương m, n là kích thước của lưới nền (m hàng, n cột) m dòng tiếp theo, dòng thứ i ghi n số nguyên không âm, số nguyên thứ j biểu thị chiều cao của cột dựng tại ô (i, j) của lưới. Các số cách nhau ít nhất một dấu cách.

Kết quả ra đặt trong file văn bản PAINT.OUT, ghi số diện tích cần sơn.

Ví dụ:

Với hình vẽ bên, các cột được xây trên nền kích thước 2×3 . Các file dữ liệu vào và kết quả ra sẽ là:

PAINT . INP		
2	3	
4	3	4
1	2	1

PAINT . OUT
42

Ý tưởng thực hiện:

**** Ý tưởng chung:**

Ta cần xét các vị trí trên, dưới, trái phải, của mỗi mỗi cột, nếu nó cao hơn các vị trí xung quanh bao nhiêu thì cần sơn bấy nhiêu.

Và cần tính cả mặt trên của cột nên giá trị được sơn cần phải cộng thêm số cột.

**** Cụ thể hóa ý tưởng:**

B1: Đầu tiên ta cần đọc file PAINT.INP để lấy được thông tin về lưới cột.

Trong quá trình này cần tạo một đường viền bao xung quanh các cột với giá trị là 0 để tính cho trường hợp các cột ở biên.

Ở hàm đọc file (void readFile) đã cho vòng lặp chạy đến m+2 và n+2 để tạo biên, dòng if số 12 để kiểm tra biến vị trí có đang ở viền hay không và gán giá trị 0 hoặc đọc giá trị từ file nếu không nằm ở viền bao quanh.

B2: Thao tác xử lý (void paint) bắt đầu từ việc gán biến tổng s=m*n với mục đích sơn trước các phần đỉnh cột.

Kế đó, hai vòng lặp for lồng nhau chạy từ i=1, j=1 đến lần lượt m+1 và n+1 với mục đích duyệt qua mảng và bỏ qua phần đường viền đã tạo ở hàm đọc file phía trên. Lúc này các câu lệnh điều kiện thực hiện nhiệm vụ kiểm tra và xử lý:

+ Nếu bên trái có cột cao hơn thì lấy cột trái trừ cột này, ta sẽ được số mặt bên trái của ô này.

+ Nếu bên phải có cột cao hơn thì lấy cột phải trừ cột này, ta sẽ được số mặt bên phải của ô này.

+ Làm tương tự với trước sau... như thế sẽ tìm đc số ô cần sơn bên mặt của tất cả các ô. Kết quả cộng dồn vào biến s (đã được sơn đỉnh cột từ đầu).

B3: Kết quả được tính sẽ được ghi vào file PAINT.OUT

Code minh họa:

```
#include <iostream>

void readFile(int& m, int& n, int arr[1000][1000])
{
    FILE* f = fopen("PAINT.INP", "rt");
    fscanf(f, "%d", &m);
    fscanf(f, "%d", &n);
    for (int i = 0; i < m + 2; i++) {
        for (int j = 0; j < n + 2; j++) {
            if (i == 0 || i == m + 1 || j == 0 || j == n + 1)
                arr[i][j] = 0;
            else
                fscanf(f, "%d", &arr[i][j]);
        }
    }
    fclose(f);
}

void paint(int m, int n, int arr[1000][1000])
{
    int s = m * n;
```

```

    for (int i = 1; i < m + 1; i++) {
        for (int j = 1; j < n + 1; j++) {
            if (arr[i][j] > arr[i - 1][j])
                s += (arr[i][j] - arr[i - 1][j]);
            if (arr[i][j] > arr[i + 1][j])
                s += (arr[i][j] - arr[i + 1][j]);
            if (arr[i][j] > arr[i][j - 1])
                s += (arr[i][j] - arr[i][j - 1]);
            if (arr[i][j] > arr[i][j + 1])
                s += (arr[i][j] - arr[i][j + 1]);
        }
    }
    FILE* f = fopen("PAINT.OUT", "wt");
    fprintf(f, "%d", s);
    fclose(f);
}

int main()
{
    int m, n, arr[1000][1000];
    std::cout << "Bai 029 running...\n";
    readFile(m, n, arr);
    paint(m, n, arr);
    std::cout << "Success...\n";
    return 0;
}

```

045. MÃ GRAY

- **Bài toán:**

Một hình tròn được chia làm $2n$ hình quạt đồng tâm, các hình quạt được đánh số từ 1 tới 2^n theo chiều kim đồng hồ. Hãy chỉ ra **một** cách xếp tất cả số từ 0 tới $2^n - 1$ vào các hình quạt, mỗi số vào một hình quạt sao cho bất cứ hai số nào ở hai hình quạt cạnh nhau đều chỉ khác nhau đúng 1 bit trong biểu diễn nhị phân của nó.

Ví dụ: Với $n = 4$:

0 = 0000

1 = 0001

2 = 0010

3 = 0011

4 = 0100

5 = 0101

6 = 0110

7 = 0111

8 = 1000

9 = 1001

10 = 1010

11 = 1011

12 = 1100

13 = 1101

14 = 1110

15 = 1111

Dữ liệu: Nhập từ bàn phím số nguyên dương n . Giới hạn ($1 \leq n \leq 20$).

Kết quả: Ghi ra File (of LongInt) GRAYCODE.OUT gồm 2^n số nguyên kiểu LongInt theo đúng thứ tự từ số ghi trên hình quạt 1 tới số ghi trên hình quạt 2^n .

- **Ý tưởng:**

Với mỗi số n ta sẽ phải biểu diễn cơ số 2 của số từ 1 đến 2^n nhưng thứ tự của nó không phải tăng dần mà có sự đối xứng, chúng ta tìm ra quy luật đối xứng của nó

$A[1]=0$ đối xứng với $A[n]=2^n$

$A[2]=1$ đối xứng là $A[n-1]=2^n$

Ta có quy tắc tính toán

Quy ước: với $n = 1$ là Gray thứ 1, $n = 2$ là Gray thứ 2

Gray thứ 2 là sao chép và đảo ngược của Gray thứ nhất, nửa đầu thêm 0 và nửa sau thêm 1

Như vậy muốn tìm Gray thứ n ta phải tìm Gray thứ $n-1$.

Việc còn lại là biểu diễn nhị phân của dãy theo độ dài n là xong

- **Code minh họa**

```

#include <bits/stdc++.h>
#include <fstream>

using namespace std;

int main(){
    string s[10000];

    int n;

    cin>>n;

    int k =pow(2,n);

    s[1]="0";
    s[2]="1";

    if(n>1){
        int t =2;

        for(int i = 2; i<=n; i++){
            int l=pow(2,i);

            for(int j= 1; j<=t; j++){
                s[l-j+1] = "1" + s[j];

                s[j] = "0" + s[j];
            }

            t= l;
        }
    }

    ofstream f("GRAYCODE.OUT");

    for(int j = 1; j<=k; j++){
        cout<<s[j]<<" ";

        f<< s[j]<<" ";
    }

    f.close();
}

```

054: THỨ TỰ TỪ ĐIỂN.

1. Đề bài:

Một bảng danh mục gồm các từ đã được sắp xếp theo một trật tự từ điển nào đấy (không nhất thiết là từ điển thông thường). Yêu cầu từ bảng danh mục, hãy khôi phục lại trật tự từ điển đã dùng.





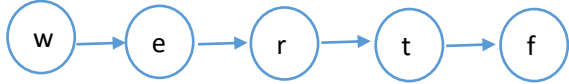
Dữ liệu vào được cho bởi file văn bản NOTE.INP. Dòng đầu là số lượng từ, các dòng tiếp, (theo thứ tự) mỗi dòng là một từ trong bảng danh mục. Giả thiết rằng mỗi từ đều không quá 20 ký tự được lấy trong bảng chữ cái nhỏ tiếng Anh (từ 'a' đến 'z'). Số lượng từ trong bảng danh mục không quá 10000.

- **Kết quả** đưa ra file văn bản NOTE.OUT gồm một dòng là xâu gồm các chữ cái đã xuất hiện trong bảng danh mục. Các chữ cái trong xâu viết liền nhau và theo thứ tự phù hợp với trật tự từ điển đã dùng.

NOTE.INP	NOTE.OUT
10 svxngqqnsnv qv snngg qsqsqvgsqq qqns qnvq nsxnxnvsqsvv s nqg nn xsgvsgggqvsgqsgv xxgxxggsvnxsxsn qq	gsvqnx

2. Ý tưởng:

- Thuật toán: dựa trên bài toán Alien Dictionary và thuật toán Topological Sorting
- Ý tưởng: Ý tưởng là tạo một đồ thị các ký tự và sau đó tìm cách Topological Sorting của đồ thị đã tạo.
- Ví dụ:
 - Input: [wrt , wrf , er, ett , rftt]
 - Output: wertf
 - Ta xét các cặp từ trong Input để tìm ra thứ tự từ điển như sau:

STT	Xét cặp từ	Kết quả
1	wrt , wrf Ta thấy: - 2 kí tự đầu wr trùng nhau. - kí tự cuối t và f khác nhau. - và t đứng trước f	 <pre> graph LR t((t)) --> f((f)) </pre>
2	wrf , er Ta thấy: - w khác e và w đứng trước e	 <pre> graph LR w((w)) --> e((e)) </pre>
3	er, ett Ta thấy: - kí tự đầu e trùng nhau. - kí tự tiếp theo r và t khác nhau. - và r đứng trước t	 <pre> graph LR r((r)) --> t((t)) </pre>
4	ett , rftt Ta thấy: - e khác r và e đứng trước r - t khác f và t đứng trước f(đã xét) - kí tự cuối t trùng nhau.	 <pre> graph LR e((e)) --> r((r)) </pre>
	Từ 1, 2, 3, 4, 5	 <pre> graph LR w((w)) --> e((e)) e --> r((r)) r --> t((t)) t --> f((f)) </pre>

3. Mã giả

- Chuẩn bị

- + Một đồ thị degree
- + Xác định một bản đồ được gọi là đồ thị
- + n = kích thước của từ

For khởi tạo $i = 0$, $i < \text{size of words}$, tăng i lên 1, do

For khởi tạo $j = 0$, khi $j < \text{size of words}[i]$, tăng j lên 1), do

$\text{degree}[\text{words}[i, j]] = 0$

For $i=0$, khi $i < n - 1$, tăng i lên 1, do

$i := \text{minimum of size of words}[i] \text{ and size of words}[i + 1]$

For $j := 0$, khi $j < i$, tăng j lên 1, do

$x := \text{words}[i, j]$

$y := \text{words}[i + 1, j]$

if x không bằng y , then

1. chèn y vào cuối $\text{graph}[x]$
2. (tăng $\text{degree}[y]$ lên 1)
3. Thoát vòng lặp

$\text{ret} := \text{chuỗi trống}$

Define queue q

For mỗi cặp key-value trong, hãy thực hiện -

If giá trị của nó bằng 0, then—

insert key vào q

while (not q is empty), do

$x = \text{phần tử đầu tiên của } q$

xóa phần tử khỏi q

$\text{ret} := \text{ret} + x$

for each sit trong graph do

decrease $\text{degree}[\text{sit}]$ by 1

if $\text{degree}[\text{sit}] = 0$, then —

insert sit into q

(increase sit by 1)

return (if size of ret = size of degree thì return ret, nếu không return chuỗi trống).

4. Code

```
#include <bits/stdc++.h>
using namespace std;
int n;
vector<string> v;
void docFile()
{
    ifstream ifs("NOTE.INP");

    if (!ifs) {
        cerr << "Error: file not opened." << endl;
        return;
    }

    string str;

    ifs >> str;
    while (ifs >> str) {
        // cout << str << endl;
        v.push_back(str);
    }
}

void ghiFile(string s)
{
    ofstream f2("NOTE.OUT");
    f2 << s;
    f2.close();
}

string alienOrder(vector<string>& words)
{
    map<char, int> degree;
    map<char, vector<char> > graph;
    int n = words.size();
    for (int i = 0; i < words.size(); i++) {
        for (int j = 0; j < words[i].size(); j++) {
            degree[words[i][j]] = 0;
        }
    }
    for (int i = 0; i < n - 1; i++) {
        int l = min((int)words[i].size(), (int)words[i + 1].size());
        for (int j = 0; j < l; j++) {
            char x = words[i][j];
            char y = words[i + 1][j];
            if (x != y) {
                graph[x].push_back(y);
                degree[y]++;
                break;
            }
        }
    }
    string ret = "";
```

```

queue<char> q;
map<char, int>::iterator it = degree.begin();
while (it != degree.end()) {
    if (it->second == 0) {
        q.push(it->first);
    }
    it++;
}
while (!q.empty()) {
    char x = q.front();
    q.pop();
    ret += x;
    vector<char>::iterator sit = graph[x].begin();
    while (sit != graph[x].end()) {
        degree[*sit]--;
        if (degree[*sit] == 0) {
            q.push(*sit);
        }
        sit++;
    }
}
return ret.size() == degree.size() ? ret : "";
}

int main()
{
    //vector<string> v = { "svxngqqnsnvqv", "svxngqqnsnvqv",
    "svxngqqnsnvqv", "qqns", "qnvq", "nsxnxnvsqsvvs", "nqg", "nn",
    "xsgvsgggqvsqqsxgv", "xxgxxggsvnxsnsnqq" };
    docFile();
    cout << alienOrder(v);
    ghiFile(alienOrder(v));
}

```