Table of Contents

[BÀI THỰC HÀNH SỐ 3 – TUẦN 13 2](#_Toc183973245)

[BÀI TẬP TRÊN LAB 2](#_Toc183973246)

[Bài 3.1: Dãy Lucas được định nghĩa bởi Ln = Ln-1 + Ln-2 với L0 = 2, L1 = 1. Hãy viết hàm tính số Lucas thứ n. 2](#_Toc183973247)

[Bài 3.2: Trên bàn cờ vua kích thước n\*n có một quân mã đang ở ô (1, 1). Hãy đưa ra một dãy các di chuyển của mã sao cho mỗi ô trên bàn cờ đều được đi qua đúng 1 lần (ô (1, 1) được xem là đã đi qua). 3](#_Toc183973248)

[Bài 3.3: Một người xuất phát tại thành phố 1, muốn đi thăm tất cả các thành phố khác, mỗi thành phố đúng 1 lần và quay về 1. Chi phí để đi từ thành phố i sang thành phố j là cij. Hãy tìm tổng chi phí nhỏ nhất có thể. 7](#_Toc183973249)

[Bài 3.4: Cho dãy a có n phần tử. Một dãy con của a là dãy thu được bằng cách xóa đi một số phần tử của a và giữ nguyên thứ tự các phần tử còn lại (có thể không xóa phần tử nào). Hãy tìm dãy con tăng dài nhất của a. 11](#_Toc183973250)

[Bài 3.5: Tính hệ số tổ hợp C(n, k). 13](#_Toc183973251)

[Bài 3.6: Tìm ước chung lớn nhất của hai số nguyên a, b cho trước. 16](#_Toc183973252)

[Bài 3.7: Sử dụng phương pháp khử đệ quy bằng stack, hãy liệt kê các xâu nhị phân độ dài n không có k bit 1 nào liên tiếp. 18](#_Toc183973253)

[Bài 3.8: Bạn đang muốn kiểm tra xem một vật cho trước có đúng nặng MM như người ta nói hay không. Có một cân thăng bằng và nn quả cân. Quả thứ ii nặng mimi. Hãy chỉ ra một cách cân thỏa mãn. Quy cách in ra đã được tích hợp trong mã nguồn dưới. 22](#_Toc183973254)

[BÀI TẬP VỀ NHÀ 25](#_Toc183973255)

[Bài 3.9: Một y tá cần lập lịch làm việc trong N ngày, mỗi ngày chỉ có thể là làm việc hay nghỉ ngơi. Một lịch làm việc là tốt nếu không có hai ngày nghỉ nào liên tiếp và mọi chuỗi ngày tối đại làm việc liên tiếp đều có số ngày thuộc đoạn [K1,K2] . Hãy liệt kê tất cả các cách lập lịch tốt, với mỗi lịch in ra trên một dòng một xâu nhị phân độ dài n với bit 0/1 tương ứng là nghỉ/làm việc. Các xâu phải được in ra theo thứ tự từ điển. 25](#_Toc183973256)

[Bài 3.10: Khoảng cách Hamming giữa hai xâu cùng độ dài là số vị trí mà ký tự tại vị trí đó là khác nhau trên hai xâu. Cho S là xâu gồm n ký tự 0. Hãy liệt kê tất cả các xâu nhị phân độ dài n , có khoảng cách Hamming với S bằng H . Các xâu phải được liệt kê theo thứ tự từ điển. 28](#_Toc183973257)

[Bài 3.11: Superior là một hòn đảo tuyệt đẹp với n địa điểm chụp ảnh và các đường một chiều nối các điểm chụp ảnh với nhau. Đoàn khách tham quan có r người với sở thích chụp ảnh khác nhau. Theo đó, mỗi người sẽ đưa ra danh sách các địa điểm mà họ muốn chụp. Bạn cần giúp mỗi người trong đoàn lập lịch di chuyển sao cho đi qua các điểm họ yêu cầu đúng một lần, không đi qua điểm nào khác, bắt đầu tại điểm đầu tiên và kết thúc tại điểm cuối cùng trong danh sách mà họ đưa ra, và có tổng khoảng cách đi lại là nhỏ nhất. 31](#_Toc183973258)

[Bài 3.12: Cho đồ thị vô hướng G , hãy đếm số đường đi đi qua k cạnh và không đi qua đỉnh nào quá một lần. 36](#_Toc183973259)

# BÀI THỰC HÀNH SỐ 3 – TUẦN 13

## BÀI TẬP TRÊN LAB

### Bài 3.1: Dãy Lucas được định nghĩa bởi Ln = Ln-1 + Ln-2 với L0 = 2, L1 = 1. Hãy viết hàm tính số Lucas thứ n.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

#include<bits/stdc++.h>

using namespace std;

int lucas(int *n*) {

    if (*n* == 0) return 2;

    if (*n* == 1) return 1;

    return lucas(*n* - 1) + lucas(*n* - 2);

}

int main() {

    cout << lucas(5) << endl;

    return 0;

}

### Bài 3.2: Trên bàn cờ vua kích thước n\*n có một quân mã đang ở ô (1, 1). Hãy đưa ra một dãy các di chuyển của mã sao cho mỗi ô trên bàn cờ đều được đi qua đúng 1 lần (ô (1, 1) được xem là đã đi qua).

A white background with blue and green colors

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated

A white rectangular object with a black border

Description automatically generated

A green square with pink border

Description automatically generated

A green wall with a white edge

Description automatically generated

#include <iostream>

#include <vector>

using namespace std;

// Các bước di chuyển của quân mã

const int hx[] = {1, 1, 2, 2, -1, -1, -2, -2};

const int hy[] = {2, -2, 1, -1, 2, -2, 1, -1};

int n;                      // Kích thước bàn cờ

int mark[100][100];         // Mảng đánh dấu ô đã đi qua

int X[100 \* 100], Y[100 \* 100]; // Vị trí các bước đi

// Hàm in kết quả

void print\_solution() {

    for (int i = 1; i <= n \* n; ++i) {

        printf("(%d %d)\n", X[i], Y[i]);

    }

    exit(0); // Kết thúc chương trình sau khi tìm được một lời giải

}

// Hàm thử nghiệm các bước đi của quân mã

void TRY(int *k*) {

    for (int i = 0; i < 8; ++i) {

        int xx = X[*k* - 1] + hx[i]; // Tính tọa độ mới

        int yy = Y[*k* - 1] + hy[i];

        // Kiểm tra bước đi có hợp lệ hay không

        if (xx >= 1 && xx <= n && yy >= 1 && yy <= n && !mark[xx][yy]) {

            X[*k*] = xx; // Lưu tọa độ bước đi

            Y[*k*] = yy;

            mark[xx][yy] = 1; // Đánh dấu ô đã đi qua

            if (*k* == n \* n) { // Nếu đã đi đủ n \* n ô

                print\_solution();

            } else {

                TRY(*k* + 1); // Thử bước đi tiếp theo

            }

            // Quay lui

            mark[xx][yy] = 0;

        }

    }

}

int main() {

    cin >> n;

    // Khởi tạo bước đi đầu tiên

    X[1] = 1;

    Y[1] = 1;

    mark[1][1] = 1;

    // Bắt đầu thử nghiệm

    TRY(2);

    // Nếu không tìm được lời giải

    cout << "No solution found!" << endl;

    return 0;

}

### Bài 3.3: Một người xuất phát tại thành phố 1, muốn đi thăm tất cả các thành phố khác, mỗi thành phố đúng 1 lần và quay về 1. Chi phí để đi từ thành phố i sang thành phố j là cij. Hãy tìm tổng chi phí nhỏ nhất có thể.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A green rectangle with white border

Description automatically generated

#include <iostream>

#include <vector>

#include <climits>

using namespace std;

int n; // Số lượng thành phố

vector<vector<int>> c; // Ma trận chi phí

vector<bool> visited; // Mảng đánh dấu các thành phố đã đi qua

int min\_cost = INT\_MAX; // Chi phí nhỏ nhất

int current\_cost = 0; // Chi phí tạm thời

vector<int> path; // Đường đi tạm thời

// Hàm quay lui để thử tất cả các cách đi

void TRY(int city, int count) {

if (count == n) { // Đã đi qua tất cả các thành phố

if (c[city][0] != 0) { // Kiểm tra chi phí quay về thành phố 1

min\_cost = min(min\_cost, current\_cost + c[city][0]);

}

return;

}

for (int next\_city = 0; next\_city < n; ++next\_city) {

if (!visited[next\_city] && c[city][next\_city] != 0) { // Nếu thành phố chưa được thăm và có đường đi

visited[next\_city] = true; // Đánh dấu thành phố đã thăm

current\_cost += c[city][next\_city]; // Cộng chi phí di chuyển

if (current\_cost < min\_cost) { // Chỉ tiếp tục nếu chi phí tạm thời < min\_cost

TRY(next\_city, count + 1);

}

// Quay lui

visited[next\_city] = false;

current\_cost -= c[city][next\_city];

}

}

}

int main() {

cin >> n;

// Khởi tạo ma trận chi phí

c.resize(n, vector<int>(n));

visited.resize(n, false);

for (int i = 0; i < n; ++i) {

for (int j = 0; j < n; ++j) {

cin >> c[i][j];

}

}

// Bắt đầu từ thành phố 1

visited[0] = true;

TRY(0, 1);

// Xuất kết quả

cout << min\_cost << endl;

return 0;

}

### Bài 3.4: Cho dãy a có n phần tử. Một dãy con của a là dãy thu được bằng cách xóa đi một số phần tử của a và giữ nguyên thứ tự các phần tử còn lại (có thể không xóa phần tử nào). Hãy tìm dãy con tăng dài nhất của a.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

#include <iostream>

#include <vector>

using namespace std;

int main() {

int n;

cin >> n;

vector<int> a(n);

for (int i = 0; i < n; ++i) {

cin >> a[i];

}

// Mảng dp lưu độ dài LIS kết thúc tại mỗi phần tử

vector<int> dp(n, 1);

// Mảng parent lưu chỉ số phần tử trước đó trong LIS

vector<int> parent(n, -1);

int max\_length = 1; // Độ dài LIS dài nhất

int last\_index = 0; // Chỉ số phần tử cuối cùng trong LIS dài nhất

// Tính toán LIS

for (int i = 1; i < n; ++i) {

for (int j = 0; j < i; ++j) {

if (a[i] > a[j] && dp[i] < dp[j] + 1) {

dp[i] = dp[j] + 1;

parent[i] = j; // Lưu chỉ số phần tử trước đó trong dãy LIS

}

}

// Cập nhật độ dài LIS và chỉ số phần tử cuối cùng nếu tìm thấy LIS dài hơn

if (dp[i] > max\_length) {

max\_length = dp[i];

last\_index = i;

}

}

// Phục hồi dãy con tăng dài nhất

vector<int> lis;

while (last\_index != -1) {

lis.push\_back(a[last\_index]);

last\_index = parent[last\_index];

}

// In kết quả

cout << max\_length << endl;

for (int i = lis.size() - 1; i >= 0; --i) {

cout << lis[i] << " ";

}

cout << endl;

return 0;

}

### Bài 3.5: Tính hệ số tổ hợp C(n, k).

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

#include <iostream>

#include <vector>

using namespace std;

int main() {

    int n;

    cin >> n;

    // Mảng 2 chiều để lưu hệ số tổ hợp C(n, k)

    vector<vector<int>> C(n + 1, vector<int>(n + 1, 0));

    // Khởi tạo mảng Pascal

    for (int i = 0; i <= n; ++i) {

        C[i][0] = C[i][i] = 1; // C(n, 0) và C(n, n) đều bằng 1

    }

    // Tính các giá trị còn lại trong mảng Pascal

    for (int i = 1; i <= n; ++i) {

        for (int j = 1; j < i; ++j) {

            C[i][j] = C[i - 1][j - 1] + C[i - 1][j];

        }

    }

    // In kết quả theo định dạng yêu cầu

    for (int i = 1; i <= n; ++i) {

        for (int j = 0; j <= i; ++j) {

            cout << C[i][j] << " ";

        }

        cout << endl;

    }

        for (int i = 1; i <= n; ++i) {

        for (int j = 0; j <= i; ++j) {

            cout << C[i][j] << " ";

        }

        cout << endl;

    }

    return 0;

}

### Bài 3.6: Tìm ước chung lớn nhất của hai số nguyên a, b cho trước.

A screenshot of a computer

Description automatically generated

#include <iostream>

using namespace std;

// Hàm tính ước chung lớn nhất (gcd) bằng thuật toán Euclid

int gcd(int *a*, int *b*) {

    while (*b* != 0) {

        int temp = *a* % *b*;

*a* = *b*;

*b* = temp;

    }

    return *a*;

}

int main() {

    int a, b;

    cin >> a >> b;  // Nhập hai số a và b

    // Tính và in ra ước chung lớn nhất

    cout << gcd(a, b) << endl;

    cout << gcd(a, b) << endl;

    return 0;

}

### Bài 3.7: Sử dụng phương pháp khử đệ quy bằng stack, hãy liệt kê các xâu nhị phân độ dài n không có k bit 1 nào liên tiếp.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generatedA green and white background

Description automatically generated

#include <bits/stdc++.h>

using namespace std;

struct *state* {

    int i, j, old\_L;

    // Constructor

    state(int *i* = 0, int *j* = 0, int *\_L* = 0) : i(*i*), j(*j*), old\_L(*\_L*) {}

};

int main() {

    int n, k;

    cin >> n >> k;

    int x[n + 1]; // Mảng lưu xâu nhị phân

    stack<*state*> s;

    // Số bit 1 liên tiếp

    int L = 0;

    s.push(*state*(1, 0, 0));

    while (!s.empty()) {

*state* &top = s.top();

        // Nếu đã có xâu nhị phân đầy đủ

        if (top.i > n) {

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

                cout << x[i] << " ";  // In ra xâu nhị phân

            cout << endl;

            s.pop();

            continue;

        }

        // Quay lui nếu cần thiết

        if (top.j > 0)

            L = top.old\_L;

        // Thăm các nhánh con của cây tìm kiếm

        if (top.j > 1) {

            s.pop();

            continue;

        }

        // Nếu thêm bit 1 hợp lệ (không vượt quá k bit 1 liên tiếp)

        if (L + 1 < k || top.j == 0) {

            x[top.i] = top.j;

            top.old\_L = L;

            L = (top.j ? L + 1 : 0);

            s.push(*state*(top.i + 1, 0, L));  // Tiến sang phần tử tiếp theo

        }

        ++top.j;

    }

    return 0;

}

### Bài 3.8: Bạn đang muốn kiểm tra xem một vật cho trước có đúng nặng MM như người ta nói hay không. Có một cân thăng bằng và nn quả cân. Quả thứ ii nặng mimi. Hãy chỉ ra một cách cân thỏa mãn. Quy cách in ra đã được tích hợp trong mã nguồn dưới.

A white background with black text

Description automatically generated

#include <bits/stdc++.h>

using namespace std;

struct *state* {

    int i, j;

    state(int *\_i* = 0, int *\_j* = 0): i(*\_i*), j(*\_j*) {}

};

int main() {

    int n, M;

    cin >> n >> M;

    int m[n+1];

    for (int i = 1; i <= n; ++i) cin >> m[i];

    int x[n+1];

    stack<*state*> s;

    //# sum of selected weights

    int sum = 0;

    s.push(*state*(1, -1));

    while (!s.empty()) {

*state* &top = s.top();

        if (top.i > n) {

            if (sum == M) {

                for (int i = 1; i <= n; ++i) {

                    if (x[i] == -1) cout << '-' << m[i];

                    if (x[i] == 1) cout << '+' << m[i];

                }

                cout << "=" << M;

                return 0;

            }

            s.pop();

            continue;

        }

        // # backtrack

        if (top.j > -1) {

            sum -= m[top.i] \* x[top.i];

        }

        // # every subtree is visited

        if (top.j > 1) {

            s.pop();

            continue;

        }

        x[top.i] = top.j;

        sum += m[top.i] \* x[top.i];

        s.push(*state*(top.i + 1, -1));

        ++top.j;

    }

    cout << -1;

    return 0;

}

A screen shot of a computer

Description automatically generated

## BÀI TẬP VỀ NHÀ

### Bài 3.9: Một y tá cần lập lịch làm việc trong N ngày, mỗi ngày chỉ có thể là làm việc hay nghỉ ngơi. Một lịch làm việc là tốt nếu không có hai ngày nghỉ nào liên tiếp và mọi chuỗi ngày tối đại làm việc liên tiếp đều có số ngày thuộc đoạn [K1,K2] . Hãy liệt kê tất cả các cách lập lịch tốt, với mỗi lịch in ra trên một dòng một xâu nhị phân độ dài n với bit 0/1 tương ứng là nghỉ/làm việc. Các xâu phải được in ra theo thứ tự từ điển.

A white background with black text

Description automatically generated

#include <bits/stdc++.h>

using namespace std;

struct *State* {

    int i;               // Vị trí hiện tại trong chuỗi

    int currentLength;   // Độ dài chuỗi làm việc liên tiếp hiện tại

    int lastBit;         // Bit cuối cùng (0: nghỉ, 1: làm việc)

*string* schedule;     // Chuỗi lịch làm việc

    State(int *\_i*, int *\_currentLength*, int *\_lastBit*, *string* *\_schedule*)

        : i(*\_i*), currentLength(*\_currentLength*), lastBit(*\_lastBit*), schedule(*\_schedule*) {}

};

int main() {

    int N, K1, K2;

    cin >> N >> K1 >> K2;

    queue<*State*> q;

    vector<*string*> results;

    // Khởi tạo trạng thái ban đầu

    q.emplace(0, 0, -1, "");  // -1: chưa quyết định (trạng thái khởi đầu)

    while (!q.empty()) {

*State* top = q.front();

        q.pop();

        // Nếu đã hoàn thành chuỗi lịch làm việc

        if (top.i == N) {

            results.push\_back(top.schedule);

            continue;

        }

        // Thử đặt nghỉ (0)

        if (top.lastBit != 0) { // Không cho phép hai ngày nghỉ liên tiếp

            q.emplace(top.i + 1, 0, 0, top.schedule + "0");

        }

        // Thử đặt làm việc (1)

        if (top.currentLength + 1 <= K2) { // Không vượt quá K2 ngày làm việc liên tiếp

            q.emplace(top.i + 1, top.currentLength + 1, 1, top.schedule + "1");

        }

    }

    // Lọc kết quả theo điều kiện độ dài chuỗi làm việc liên tiếp trong [K1, K2]

    for (const *string* &schedule : results) {

        bool isValid = true;

        int workLength = 0;

        for (char c : schedule) {

            if (c == '1') {

                workLength++;

            } else {

                if (workLength > 0 && (workLength < K1 || workLength > K2)) {

                    isValid = false;

                    break;

                }

                workLength = 0;

            }

        }

        if (workLength > 0 && (workLength < K1 || workLength > K2)) {

            isValid = false;

        }

        if (isValid) {

            cout << schedule << endl;

        }

    }

    return 0;

}

A screen shot of a computer

Description automatically generated

### Bài 3.10: Khoảng cách Hamming giữa hai xâu cùng độ dài là số vị trí mà ký tự tại vị trí đó là khác nhau trên hai xâu. Cho S là xâu gồm n ký tự 0. Hãy liệt kê tất cả các xâu nhị phân độ dài n , có khoảng cách Hamming với S bằng H . Các xâu phải được liệt kê theo thứ tự từ điển.

A white background with black text

Description automatically generated

#include <bits/stdc++.h>

using namespace std;

// Hàm quay lui để liệt kê các xâu nhị phân có độ dài N và đúng H bit 1

void generateStrings(int *idx*, int *H*, int *N*, *string* &*current*, vector<*string*> &*result*) {

    if (*idx* == *N*) {

        if (*H* == 0) { // Nếu đủ H bit 1

*result*.push\_back(*current*);

        }

        return;

    }

    // Gắn '0' vào vị trí idx

*current*[*idx*] = '0';

    generateStrings(*idx* + 1, *H*, *N*, *current*, *result*);

    // Gắn '1' vào vị trí idx (nếu vẫn còn cần thêm bit 1)

    if (*H* > 0) {

*current*[*idx*] = '1';

        generateStrings(*idx* + 1, *H* - 1, *N*, *current*, *result*);

    }

}

int main() {

    int T; // Số testcase

    cin >> T;

    while (T--) {

        int N, H;

        cin >> N >> H;

        vector<*string*> result;

*string* current(N, '0'); // Xâu hiện tại độ dài N toàn '0'

        // Gọi hàm quay lui để sinh các xâu nhị phân

        generateStrings(0, H, N, current, result);

        // In kết quả

        for (const *string* &s : result) {

            cout << s << endl;

        }

        // In dòng trống giữa các testcase

        if (T > 0) {

            cout << endl;

        }

    }

    return 0;

}

A computer screen shot

Description automatically generated

### Bài 3.11: Superior là một hòn đảo tuyệt đẹp với n địa điểm chụp ảnh và các đường một chiều nối các điểm chụp ảnh với nhau. Đoàn khách tham quan có r người với sở thích chụp ảnh khác nhau. Theo đó, mỗi người sẽ đưa ra danh sách các địa điểm mà họ muốn chụp. Bạn cần giúp mỗi người trong đoàn lập lịch di chuyển sao cho đi qua các điểm họ yêu cầu đúng một lần, không đi qua điểm nào khác, bắt đầu tại điểm đầu tiên và kết thúc tại điểm cuối cùng trong danh sách mà họ đưa ra, và có tổng khoảng cách đi lại là nhỏ nhất.

A close-up of a computer screen

Description automatically generated

#include <bits/stdc++.h>

using namespace std;

const int MAX = 10000;

int n, r;  // Số đỉnh và số khách hàng

int cost[MAX][MAX];  // Ma trận chi phí

int path[MAX];  // Mảng lưu các điểm trong đường đi

bool visited[MAX];  // Mảng đánh dấu các điểm đã thăm

vector<int> destinations;  // Danh sách các điểm cần thăm

int min\_cost, total\_cost;  // Biến lưu chi phí nhỏ nhất và tổng chi phí

int start\_point, end\_point, total\_points;  // Điểm bắt đầu, điểm kết thúc, số điểm cần đi qua

bool isValid(int *prev\_index*, int *curr\_index*)

{

    if (visited[destinations[*curr\_index*]])

        return false;

    if (cost[path[*prev\_index* - 1]][destinations[*curr\_index*]] == 0)

        return false;

    return true;

}

void updateSolution()

{

    if (cost[path[total\_points - 2]][end\_point] == 0)

        return;

    min\_cost = min(min\_cost, total\_cost + cost[path[total\_points - 2]][end\_point]);

}

void findPaths(int *index*)

{

    for (int i = 1; i < total\_points - 1; i++)

    {

        if (isValid(*index*, i))

        {

            visited[destinations[i]] = true;

            total\_cost += cost[path[*index* - 1]][destinations[i]];

            path[*index*] = destinations[i];

            if (*index* == total\_points - 2)

                updateSolution();

            else

                findPaths(*index* + 1);

            visited[destinations[i]] = false;

            total\_cost -= cost[path[*index* - 1]][destinations[i]];

        }

    }

}

void processCustomerRequest(const *string*& *request*)

{

    min\_cost = INT\_MAX;

    total\_cost = 0;

    destinations.clear();

    // Phân tích yêu cầu khách hàng

*stringstream* ss(*request*);

    int point;

    while (ss >> point)

    {

        destinations.push\_back(point - 1);

    }

    start\_point = destinations[0];

    end\_point = destinations[destinations.size() - 1];

    total\_points = destinations.size();

    path[0] = start\_point;

    path[total\_points - 1] = end\_point;

    fill(visited, visited + n, false);

    findPaths(1);

    if (min\_cost == INT\_MAX)

        cout << "0" << endl;

    else

        cout << min\_cost << endl;

}

int main() {

    cin >> n >> r;

    for (int i = 0; i < n; i++)

        for (int j = 0; j < n; j++)

        {

            cin >> cost[i][j];

        }

    cin.ignore();  // Bỏ dòng trống sau input

    while (r--)

    {

*string* customer\_request;

        getline(cin, customer\_request);

        processCustomerRequest(customer\_request);

    }

    return 0;

}

// 6 3

// 0 1 2 0 1 1

// 1 0 1 1 1 0

// 0 2 0 1 3 0

// 4 3 1 0 0 0

// 0 0 1 1 0 0

// 1 0 0 0 0 0

// 1 3 5

// 6 3 2 5

// 6 1 2 3 4 5

A screen shot of a computer

Description automatically generated

### Bài 3.12: Cho đồ thị vô hướng G , hãy đếm số đường đi đi qua k cạnh và không đi qua đỉnh nào quá một lần.

A white screen with black text

Description automatically generated

#include <iostream>

#include <vector>

#include <stack>

#include <tuple>

using namespace std;

int count\_paths(int *n*, int *k*, const vector<pair<int, int>>& *edges*) {

    // Xây dựng đồ thị dưới dạng danh sách kề

    vector<vector<int>> graph(*n* + 1);

    for (const auto& edge : *edges*) {

        int u = edge.first, v = edge.second;

        graph[u].push\_back(v);

        graph[v].push\_back(u);

    }

    int total\_paths = 0;

    // Duyệt từ tất cả các đỉnh

    for (int start = 1; start <= *n*; ++start) {

        vector<bool> visited(*n* + 1, false);  // Mảng kiểm tra đã thăm đỉnh nào chưa

        stack<tuple<int, int, vector<bool>>> stk;  // Lưu trữ trạng thái: (đỉnh, độ dài, visited)

        stk.push(make\_tuple(start, 0, visited));

        while (!stk.empty()) {

            auto [v, path\_length, visited\_state] = stk.top();

            stk.pop();

            // Nếu độ dài đường đi đạt k, tăng tổng số đường đi

            if (path\_length == *k*) {

                total\_paths++;

                continue;

            }

            // Đánh dấu đỉnh v đã được thăm

            visited\_state[v] = true;

            // Duyệt các đỉnh kề với v

            for (int neighbor : graph[v]) {

                if (!visited\_state[neighbor]) {  // Nếu chưa thăm đỉnh này

                    vector<bool> new\_visited\_state = visited\_state;

                    stk.push(make\_tuple(neighbor, path\_length + 1, new\_visited\_state));

                }

            }

        }

    }

    return total\_paths;

}

int main() {

    int n, k, m;  // Số đỉnh, độ dài đường đi (k), số lượng cạnh

    cin >> n >> k;

    cin >> m;

    vector<pair<int, int>> edges(m);  // Danh sách các cạnh

    // Đọc các cạnh

    for (int i = 0; i < m; ++i) {

        int u, v;

        cin >> u >> v;

        edges[i] = {u, v};

    }

    // Tính số đường đi

    int result = count\_paths(n, k, edges) / 2;

    cout << result << endl;

    return 0;

}

A computer screen shot

Description automatically generated