Câu 1:

A. Phân tích, phân rã bài toán ban đầu về bài toán con cùng dạng và có kích thước nhỏ hơn

- **Bài toán ban đầu**: Tìm số lượng tấm thẻ nhiều nhất có thể chọn sao cho:
  1. $|i - j| \leq 10$ (khoảng cách chỉ số thẻ).
  2. $|a_i - a_j| \neq 0$.
  3. $|a_i - a_j|$ là bình phương của một số tự nhiên.
- **Phân rã**:
  - Xét $dp[i]$: Số lượng tấm thẻ tối đa có thể chọn được khi tấm thẻ cuối cùng là i.
  - $dp[i]$ phụ thuộc vào các $dp[j]$ với $j < i$, nếu j thỏa mãn các điều kiện bài toán.
  - Công thức:

$$dp[i] = \max(dp[j] + 1), \text{ với } j \text{ thỏa mãn điều kiện.}$$

B. Giải bài toán con cơ sở

- **Bài toán cơ sở**:
  - Khi $i = 0$, chỉ có thể chọn tấm thẻ đầu tiên:

$$dp[0] = 1$$

C. Xây dựng công thức, tìm mối liên hệ từ bài toán con với bài toán tổng quát

- **Công thức tổng quát**:

$$dp[i] = \max(dp[j] + 1)$$

Với j thỏa mãn:

- $0 < |i - j| \leq 10$.
- $|a_i - a_j| \neq 0$.
- $|a_i - a_j|$ là bình phương của một số tự nhiên.

D. Lập bảng phương án dựa trên công thức đã tìm

- **Lập bảng**:
  - Khởi tạo mảng dp kích thước n, với $dp[i] = 1$ (mặc định mỗi tấm thẻ đều có thể chọn riêng lẻ).
  - Duyệt i từ 0 đến $n - 1$, cập nhật $dp[i]$ dựa trên các giá trị $dp[j]$ trước đó.

E. Truy vết

- **Truy vết đường đi**:
  - Lưu mảng $parent[]$ để ghi lại chỉ số j mà $dp[i]$ được cập nhật.
  - Từ chỉ số i có giá trị $dp[i]$ lớn nhất, lần ngược lại qua $parent[i]$ để tìm đường đi.

```
#include <iostream>
#include <vector>
#include <cmath>
#include <algorithm>
using namespace std;

// Hàm kiểm tra xem sự chênh lệch có phải là bình phương của một số tự nhiên hay không
bool isPerfectSquare(int x) {
    int s = sqrt(x);
    return s * s == x;
}

// Hàm tính số lượng tấm thẻ nhiều nhất có thể chọn
int maxCards(vector<int> &cards) {
    int n = cards.size();
    vector<int> dp(n, 1); // dp[i]: Số lượng tấm thẻ nhiều nhất chọn được kết thúc tại tấm thẻ i

    for (int i = 0; i < n; i++) {
        for (int j = max(0, i - 10); j < i; j++) {
            if (abs(cards[i] - cards[j]) != 0 && isPerfectSquare(abs(cards[i] - cards[j]))) {
                dp[i] = max(dp[i], dp[j] + 1);
            }
        }
```

```
    }

    return *max_element(dp.begin(), dp.end());
}

int main() {
    int n;
    cout << "Nhập số lượng tấm thẻ: ";
    cin >> n;

    vector<int> cards(n);
    cout << "Nhập giá trị của các tấm thẻ:\n";
    for (int i = 0; i < n; i++) {
        cin >> cards[i];
    }

    int result = maxCards(cards);
    cout << "Số lượng tấm thẻ nhiều nhất có thể chọn: " << result << endl;

    return 0;
}
```

câu 2:

G. Phân tích, phân rã bài toán ban đầu về bài toán con cùng dạng và có kích thước nhỏ hơn

1. **Bài toán**:
   - Tìm tổng chi phí thấp nhất để đi từ ô $(0, 0)$ đến ô (x, y).
   - Có thể di chuyển xuống ($(i + 1, j)$), sang phải ($(i, j + 1)$), và theo đường chéo ($(i + 1, j + 1)$).
2. **Phân rã**:
   - **Bài toán con**:
     - Giả sử $dp[i][j]$ là tổng chi phí thấp nhất để đi đến ô (i, j).
     - Để tính $dp[i][j]$, chỉ cần xét chi phí từ các ô liền kề:

       $$dp[i][j] = C[i][j] + \min(dp[i - 1][j], dp[i][j - 1], dp[i - 1][j - 1])$$

       (nếu các ô này tồn tại).

H. Giải bài toán con cơ sở

- **Cơ sở**:
  - $dp[0][0] = C[0][0]$ (chi phí tại ô đầu tiên).

I. Xây dựng công thức, tìm mối liên hệ từ bài toán con với bài toán tổng quát

1. **Công thức tổng quát**:

$$dp[i][j] = C[i][j] + \min(dp[i-1][j], dp[i][j-1], dp[i-1][j-1])$$

(nếu các ô $(i-1, j)$, $(i, j-1)$, $(i-1, j-1)$ tồn tại).

2. **Mục tiêu**:
   - Tìm giá trị $dp[x][y]$, trong đó x và y là tọa độ đích.

J. Lập bảng phương án dựa trên công thức đã tìm

- Tạo mảng dp có cùng kích thước với ma trận chi phí C.
- Sử dụng công thức để tính $dp[i][j]$ lần lượt từ trái sang phải, từ trên xuống dưới.

K. Truy vết

- Để truy vết đường đi:
  - Bắt đầu từ (x, y), tìm ô trước đó có giá trị nhỏ nhất trong số $dp[i-1][j]$, $dp[i][j-1]$, $dp[i-1][j-1]$.
  - Lặp lại quá trình cho đến khi về $(0, 0)$.

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

// Hàm trả về tổng chi phí thấp nhất
int minCostPath(vector<vector<int>> &C, int rows, int cols) {
    // Tạo bảng phương án
    vector<vector<int>> dp(rows, vector<int>(cols, 0));

    // Khởi tạo ô đầu tiên
    dp[0][0] = C[0][0];

    // Khởi tạo hàng đầu tiên
    for (int j = 1; j < cols; j++) {
        dp[0][j] = dp[0][j - 1] + C[0][j];
    }

    // Khởi tạo cột đầu tiên
    for (int i = 1; i < rows; i++) {
        dp[i][0] = dp[i - 1][0] + C[i][0];
    }

    // Điền các ô còn lại
    for (int i = 1; i < rows; i++) {
```

```
        for (int j = 1; j < cols; j++) {
            dp[i][j] = min({dp[i - 1][j], dp[i][j - 1], dp[i - 1][j - 1]}) + C[i][j];
        }
    }

    // Trả về chi phí thấp nhất tại ô cuối cùng
    return dp[rows - 1][cols - 1];
}

// Hàm truy vết đường đi
void tracePath(vector<vector<int>> &dp, int rows, int cols) {
    vector<pair<int, int>> path;
    int i = rows - 1, j = cols - 1;

    while (i > 0 || j > 0) {
        path.push_back({i, j});

        if (i == 0) j--;
        else if (j == 0) i--;
        else if (dp[i - 1][j - 1] <= dp[i - 1][j] && dp[i - 1][j - 1] <= dp[i][j - 1]) {
            i--;
            j--;
        } else if (dp[i - 1][j] <= dp[i][j - 1]) {
            i--;
        } else {
            j--;
        }
    }

    path.push_back({0, 0});
    reverse(path.begin(), path.end());

    cout << "Đường đi: ";
    for (auto &p : path) {
        cout << "(" << p.first << ", " << p.second << ") ";
    }
    cout << endl;
}

int main() {
    int rows, cols;
    cout << "Nhập số hàng và số cột: ";
    cin >> rows >> cols;

    vector<vector<int>> C(rows, vector<int>(cols));
    cout << "Nhập ma trận chi phí:\n";
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            cin >> C[i][j];
        }
    }

    int minCost = minCostPath(C, rows, cols);
    cout << "Chi phí thấp nhất: " << minCost << endl;

    // Lập bảng phương án
    vector<vector<int>> dp(rows, vector<int>(cols));
    dp[0][0] = C[0][0];

    for (int j = 1; j < cols; j++) dp[0][j] = dp[0][j - 1] + C[0][j];
    for (int i = 1; i < rows; i++) dp[i][0] = dp[i - 1][0] + C[i][0];

    for (int i = 1; i < rows; i++) {
```

```
        for (int j = 1; j < cols; j++) {
            dp[i][j] = min({dp[i - 1][j], dp[i][j - 1], dp[i - 1][j - 1]}) + C[i][j];
        }
    }

    tracePath(dp, rows, cols);

    return 0;
}
```