Tuần 1

Môn: **Lý thuyết đồ thị và Tổ hợp**

Đề bài a: Chứng minh rằng với mọi $n \in \mathbb{N}^+$, ta có:

$$\sum_{i=1}^{n} i = 1 + 2 + \ldots + n = \frac{n(n+1)}{2}$$

Chứng minh: Ta sẽ sử dụng phương pháp quy nạp toán học.

Bước 1: Cơ sở quy nạp.
Xét $n = 1$:

$$\sum_{i=1}^{1} i = 1 \quad \text{và} \quad \frac{1(1+1)}{2} = \frac{2}{2} = 1$$

Mệnh đề đúng với $n = 1$.

Bước 2: Giả thiết quy nạp.
Giả sử mệnh đề đúng với $n = k$, tức là:

$$\sum_{i=1}^{k} i = \frac{k(k+1)}{2}$$

Bước 3: Bước quy nạp.
Ta cần chứng minh mệnh đề đúng với $n = k + 1$:

$$\sum_{i=1}^{k+1} i = \frac{(k+1)(k+2)}{2}$$

Ta có:

$$\begin{aligned}\sum_{i=1}^{k+1} i &= \left(\sum_{i=1}^{k} i\right) + (k+1) = \frac{k(k+1)}{2} + (k+1) \\ &= \frac{k(k+1) + 2(k+1)}{2} = \frac{(k+1)(k+2)}{2}\end{aligned}$$

Mệnh đề đúng với $n = k + 1$.

Đề bài b: Chứng minh rằng với mọi $n \in \mathbb{N}^+$, ta có:

$$\sum_{i=1}^{n} i^2 = 1^2 + 2^2 + \ldots + n^2 = \frac{n(n+1)(2n+1)}{6}$$

Chứng minh: Ta sử dụng phương pháp quy nạp toán học.

Bước 1: Cơ sở quy nạp.
Với $n = 1$:

$$\sum_{i=1}^{1} i^2 = 1^2 = 1 \quad \text{và} \quad \frac{1(1+1)(2 \cdot 1 + 1)}{6} = \frac{1 \cdot 2 \cdot 3}{6} = 1$$

Mệnh đề đúng với $n = 1$.

Bước 2: Giả thiết quy nạp.
Giả sử mệnh đề đúng với $n = k$, tức là:

$$\sum_{i=1}^{k} i^2 = \frac{k(k+1)(2k+1)}{6}$$

Bước 3: Bước quy nạp.
Xét $n = k + 1$, ta cần chứng minh:

$$\sum_{i=1}^{k+1} i^2 = \frac{(k+1)(k+2)(2k+3)}{6}$$

Ta có:

$$\sum_{i=1}^{k+1} i^2 = \sum_{i=1}^{k} i^2 + (k+1)^2 = \frac{k(k+1)(2k+1)}{6} + (k+1)^2$$

$$= \frac{k(k+1)(2k+1) + 6(k+1)^2}{6} = \frac{(k+1)\left[k(2k+1) + 6(k+1)\right]}{6}$$

$$= \frac{(k+1)(2k^2+7k+6)}{6} = \frac{(k+1)(k+2)(2k+3)}{6}$$

Mệnh đề đúng với $n = k + 1$.

Kết luận: Theo nguyên lý quy nạp toán học, mệnh đề đúng với mọi $n \in \mathbb{N}^+$.

Đề bài c: Chứng minh rằng với mọi $n \in \mathbb{N}^+$, ta có:

$$\sum_{i=1}^{n} i^3 = \frac{n^2(n+1)^2}{4}$$

Chứng minh: Ta sử dụng phương pháp quy nạp.

Bước 1: Cơ sở quy nạp.
Với $n = 1$:

$$\sum_{i=1}^{1} i^3 = 1^3 = 1, \quad \frac{1^2(1+1)^2}{4} = \frac{4}{4} = 1$$

Mệnh đề đúng với $n = 1$.

Bước 2: Giả thiết quy nạp.
Giả sử mệnh đề đúng với $n = k$:

$$\sum_{i=1}^{k} i^3 = \frac{k^2(k+1)^2}{4}$$

Bước 3: Bước quy nạp.

Ta chứng minh mệnh đề đúng với $n = k + 1$:

$$\sum_{i=1}^{k+1} i^3 = \sum_{i=1}^{k} i^3 + (k+1)^3 = \frac{k^2(k+1)^2}{4} + (k+1)^3$$

$$= (k+1)^2 \left(\frac{k^2}{4} + (k+1) \right) = (k+1)^2 \cdot \frac{k^2 + 4k + 4}{4} = \frac{(k+1)^2(k+2)^2}{4}$$

Mệnh đề đúng với $n = k + 1$.

Đề bài e: Chứng minh rằng với mọi $n \in \mathbb{N}^+$, ta có:

$$\sum_{i=1}^{n} (2i - 1) = n^2$$

Chứng minh: Ta sử dụng phương pháp quy nạp.

Bước 1: Cơ sở quy nạp.
Với $n = 1$:

$$\sum_{i=1}^{1} (2i - 1) = 1, \quad 1^2 = 1 \Rightarrow \text{Mệnh đề đúng với } n = 1.$$

Bước 2: Giả thiết quy nạp.
Giả sử mệnh đề đúng với $n = k$, tức là:

$$\sum_{i=1}^{k} (2i - 1) = k^2$$

Bước 3: Bước quy nạp.
Ta cần chứng minh:

$$\sum_{i=1}^{k+1} (2i - 1) = (k+1)^2$$

Ta có:

$$\sum_{i=1}^{k+1} (2i - 1) = \left(\sum_{i=1}^{k} (2i - 1) \right) + (2(k+1) - 1) = k^2 + (2k + 1) = (k+1)^2$$

Vậy mệnh đề đúng với $n = k + 1$.

Bài g. Tính hai tổng sau:

$$\sum_{i=1}^{n} (2i - 1)^2 \quad \text{và} \quad \sum_{i=1}^{n} (2i)^3$$

1. Tổng của các số lẻ bình phương:

$$\sum_{i=1}^{n}(2i-1)^2$$

Khai triển:

$$(2i-1)^2 = 4i^2 - 4i + 1 \Rightarrow \sum_{i=1}^{n}(2i-1)^2 = 4\sum_{i=1}^{n} i^2 - 4\sum_{i=1}^{n} i + \sum_{i=1}^{n} 1$$

Áp dụng công thức:

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}, \quad \sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}, \quad \sum_{i=1}^{n} 1 = n$$

Ta có:

$$= 4 \cdot \frac{n(n+1)(2n+1)}{6} - 4 \cdot \frac{n(n+1)}{2} + n = \frac{2n(n+1)(2n+1)}{3} - 2n(n+1) + n$$

$$= \frac{2n(n+1)(2n+1) - 6n(n+1) + 3n}{3} = \frac{n(4n^2-1)}{3}$$

Vậy:

$$\sum_{i=1}^{n}(2i-1)^2 = \frac{n(4n^2-1)}{3}$$

2. Tổng lập phương của các số chẵn:

$$\sum_{i=1}^{n}(2i)^3 = 8\sum_{i=1}^{n} i^3 = 8 \cdot \left(\frac{n(n+1)}{2}\right)^2 = 2n^2(n+1)^2$$

Vậy:

$$\sum_{i=1}^{n}(2i)^3 = 2n^2(n+1)^2$$

Đề bài h: Tính tổng:

$$\sum_{i=1}^{n}(2i-1)^3$$

Lời giải:
Tổng trên là tổng lập phương của n số lẻ đầu tiên:

$$1^3 + 3^3 + 5^3 + \cdots + (2n-1)^3$$

Ta có công thức:

$$\sum_{i=1}^{n}(2i-1)^3 = n^2(2n^2-1)$$

Chứng minh:
Khai triển:

$$(2i-1)^3 = 8i^3 - 12i^2 + 6i - 1 \Rightarrow \sum_{i=1}^{n}(2i-1)^3 = 8\sum i^3 - 12\sum i^2 + 6\sum i - \sum 1$$

Thay công thức:

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}, \quad \sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}, \quad \sum_{i=1}^{n} i^3 = \left(\frac{n(n+1)}{2}\right)^2, \quad \sum 1 = n$$

Từ đó:

$$\sum_{i=1}^{n}(2i-1)^3 = 8 \cdot \left(\frac{n(n+1)}{2}\right)^2 - 12 \cdot \frac{n(n+1)(2n+1)}{6} + 6 \cdot \frac{n(n+1)}{2} - n$$

Rút gọn biểu thức trên, ta được:

$$\sum_{i=1}^{n}(2i-1)^3 = n^2(2n^2-1)$$

BÀI 1: In Pascal Triangle & Khai triển Newton

1. Tính tổ hợp và xây dựng Tam giác Pascal:

Dưới đây là đoạn mã C++ sử dụng đệ quy để tính tổ hợp $C(n,k)$ và in ra tam giác Pascal đến dòng thứ n:

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

int C(int n, int k) {
    if (k == 0 || k == n) return 1;
    return C(n - 1, k - 1) + C(n - 1, k);
}

void printPascal(int n) {
    for (int i = 0; i <= n; ++i) {
        for (int k = 0; k <= i; ++k)
            cout << C(i, k) << " ";
        cout << endl;
    }
}

void binomialExpansion(int n) {
    cout << "(a + b)^" << n << " = ";
    for (int k = 0; k <= n; ++k) {
```

```
        cout << C(n, k) << "*a^" << n - k << "*b^" << k;
        if (k < n) cout << " + ";
    }
    cout << endl;
}

int main() {
    int n;
    cout << "Nhap n: ";
    cin >> n;
    cout << "Tam giac Pascal:\n";
    printPascal(n);
    binomialExpansion(n);
    return 0;
}
```

Giải thích chi tiết chương trình

1. Hàm tính tổ hợp $C(n, k)$:

```
int C(int n, int k) {
    if (k == 0 || k == n) return 1;
    return C(n - 1, k - 1) + C(n - 1, k);
}
```

- Đây là hàm đệ quy tính tổ hợp theo định nghĩa:

$$C(n,k) = \begin{cases} 1 & \text{nếu } k = 0 \text{ hoặc } k = n \\ C(n-1,k-1) + C(n-1,k) & \text{ngược lại} \end{cases}$$

- Khi gọi `C(5, 2)`, chương trình sẽ tự động chia nhỏ thành các lời gọi con cho đến khi gặp các trường hợp cơ sở (`k == 0` hoặc `k == n`).

2. Hàm in tam giác Pascal:

```
void printPascal(int n) {
    for (int i = 0; i <= n; ++i) {
        for (int k = 0; k <= i; ++k)
            cout << C(i, k) << " ";
        cout << endl;
    }
}
```

- Hàm này in ra từng dòng của tam giác Pascal từ dòng 0 đến n.
- Với mỗi dòng i, ta in các giá trị $C(i,0), C(i,1), \ldots, C(i,i)$.
- Ví dụ nếu $n = 3$, đầu ra sẽ là:

```
1
1 1
1 2 1
1 3 3 1
```

3. Hàm khai triển nhị thức Newton $(a+b)^n$:

```
void binomialExpansion(int n) {
    cout << "(a + b)^" << n << " = ";
    for (int k = 0; k <= n; ++k) {
        cout << C(n, k) << "*a^" << n - k << "*b^" << k;
        if (k < n) cout << " + ";
    }
    cout << endl;
}
```

- Dựa theo công thức khai triển Newton:

$$(a+b)^n = \sum_{k=0}^{n} C(n,k) \cdot a^{n-k} \cdot b^k$$

- Hàm duyệt từ $k = 0$ đến n và in từng hạng tử theo đúng định dạng $C(n,k)a^{n-k}b^k$.
- Nếu $n = 2$ thì đầu ra sẽ là:

$$(a+b)^2 = 1 * a^2 * b^0 + 2 * a^1 * b^1 + 1 * a^0 * b^2$$

4. Hàm `main()`:

```
int main() {
    int n;
    cout << "Nhap n: ";
    cin >> n;
    cout << "Tam giac Pascal:\n";
    printPascal(n);
    binomialExpansion(n);
    return 0;
}
```

- Chương trình yêu cầu người dùng nhập vào một số nguyên n.
- Sau đó in ra tam giác Pascal và khai triển nhị thức Newton với số mũ n.

Bài toán 2: Tổ hợp và kiểm tra overflow

Yêu cầu: Viết chương trình bằng C/C++ để:

- Tính các giá trị: $P_n = n!$, $A_n^k = \frac{n!}{(n-k)!}$, $C_n^k = \frac{n!}{k!(n-k)!}$, và số Catalan thứ n:

$$C_n = \frac{(2n)!}{(n+1)! \cdot n!}$$

- Xác định giá trị nhỏ nhất của n tại đó xảy ra **overflow**.

Chương trình C++:

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

unsigned long long factorial(int n) {
    unsigned long long res = 1;
    for (int i = 2; i <= n; ++i) {
        if (res > ULLONG_MAX / i) {
            cout << "Overflow tại n = " << i << " khi tính giai thừa.\n";
            return 0;
        }
        res *= i;
    }
    return res;
}

unsigned long long permutation(int n, int k) {
    unsigned long long a = factorial(n);
    unsigned long long b = factorial(n - k);
    return (b == 0) ? 0 : a / b;
}

unsigned long long combination(int n, int k) {
    unsigned long long a = factorial(n);
    unsigned long long b = factorial(k);
    unsigned long long c = factorial(n - k);
    return (b == 0 || c == 0) ? 0 : a / (b * c);
}

unsigned long long catalan(int n) {
    unsigned long long a = factorial(2 * n);
    unsigned long long b = factorial(n + 1);
    unsigned long long c = factorial(n);
    return (b == 0 || c == 0) ? 0 : a / (b * c);
}
```

```
int main() {
    int n, k;
    cout << "Nhap n (n > 0): ";
    cin >> n;
    cout << "Nhap k (0 <= k <= n): ";
    cin >> k;

    cout << "\n=== Ket qua ===\n";

    unsigned long long fn = factorial(n);
    if (fn != 0) cout << "Pn = " << fn << endl;

    unsigned long long Ank = permutation(n, k);
    if (Ank != 0) cout << "A^k_n = " << Ank << endl;

    unsigned long long Cnk = combination(n, k);
    if (Cnk != 0) cout << "C^k_n = " << Cnk << endl;

    unsigned long long Catn = catalan(n);
    if (Catn != 0) cout << "Catalan(n) = " << Catn << endl;

    return 0;
}
```

Giải thích chi tiết chương trình:

- **Hàm `factorial(int n)`**: Tính $n!$ và kiểm tra nếu tại bất kỳ bước nào $res >$ `ULLONG_MAX`$/i$, tức là phép nhân tiếp theo sẽ gây tràn số. Khi đó chương trình in ra cảnh báo overflow và trả về 0.
- **Hàm `permutation(n, k)`**: Tính hoán vị $A_n^k = \frac{n!}{(n-k)!}$ bằng cách gọi `factorial()`.
- **Hàm `combination(n, k)`**: Tính tổ hợp $C_n^k = \frac{n!}{k!(n-k)!}$. Nếu mẫu số bị overflow thì trả về 0 để tránh sai kết quả.
- **Hàm `catalan(n)`**: Tính số Catalan thứ n theo công thức:

$$C_n = \frac{(2n)!}{(n+1)! \cdot n!}$$

- **Hàm `main()`**: Cho người dùng nhập n và k, sau đó hiển thị:
  - $P_n = n!$
  - A_n^k
  - C_n^k
  - Số Catalan thứ n
- Nếu bất kỳ phép toán nào gây tràn số thì chương trình sẽ in ra thông báo `Overflow tại n = ...`.

Tuần 2 - Tuần 3

Môn: **Lý thuyết đồ thị và Toán Tổ hợp**

1. Bài toán : Kiểm tra dãy số có phải là dãy bậc đồ thị (graphical sequence)

Viết chương trình C++ sử dụng thuật toán **Havel–Hakimi** để kiểm tra xem một dãy số nguyên không âm có thể là dãy bậc của một đồ thị đơn vô hướng hay không.

Ý tưởng thuật toán:

1. Xóa tất cả các phần tử 0 khỏi dãy.
2. Sắp xếp dãy theo thứ tự giảm dần.
3. Lấy phần tử đầu tiên d, xóa nó khỏi dãy.
4. Trừ 1 vào d phần tử tiếp theo. Nếu không đủ hoặc xuất hiện số âm thì không hợp lệ.
5. Lặp lại cho đến khi dãy rỗng hoặc có lỗi.

Mã nguồn chương trình:

Listing 1: Thuật toán kiểm tra graphical sequence bằng C++

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

bool is_graphical_sequence(vector<int> degrees) {
    while (true)

        degrees.erase(remove(degrees.begin(), degrees.end(), 0),
           degrees.end());

        if (degrees.empty()) return true;


        sort(degrees.begin(), degrees.end(), greater<int>());

        int d = degrees[0];
        degrees.erase(degrees.begin());

        if (d > degrees.size()) return false;

        for (int i = 0; i < d; ++i) {
            degrees[i]--;
            if (degrees[i] < 0) return false;
        }
    }
}

int main() {
    int t;
```

```
    cin >> t;

    while (t--) {
        int n;
        cin >> n;
        vector<int> degrees(n);
        for (int i = 0; i < n; ++i) {
            cin >> degrees[i];
        }

        cout << (is_graphical_sequence(degrees) ? 1 : 0) << endl;
    }

    return 0;
}
```

2. Bài toán chia kẹo Euler – Phần (d) và (e)

Cho $m, n \in \mathbb{N}$ và các số nguyên m_i, M_i (với $1 \leq i \leq n$). Xét phương trình:

$$\sum_{i=1}^{n} x_i = m$$

(d) **Đề bài:** Đếm số nghiệm nguyên của phương trình:

$$\sum_{i=1}^{n} x_i = m \quad \text{sao cho } x_i \geq m_i,\ \forall i \in [n]$$

Lời giải: Đặt $x_i' = x_i - m_i \geq 0$ Khi đó:

$$\sum_{i=1}^{n} x_i' = m - \sum_{i=1}^{n} m_i$$

Là bài toán đếm số nghiệm nguyên không âm với tổng $m - \sum m_i$. Số nghiệm:

$$\boxed{\binom{m - \sum m_i + n - 1}{n - 1}} \quad \text{nếu } m \geq \sum m_i$$

Cài đặt C++:

```
// bai_d.cpp
#include <iostream>
#include <vector>
using namespace std;

long long binomial(int n, int k) {
    if (k < 0 || k > n) return 0;
    long long res = 1;
```

```
    for (int i = 1; i <= k; ++i) {
        res *= (n - i + 1);
        res /= i;
    }
    return res;
}

long long count_solutions_lower_bound(int m, const vector<int>&
    mi) {
    int S = 0, n = mi.size();
    for (int x : mi) S += x;
    if (m < S) return 0;
    return binomial(m - S + n - 1, n - 1);
}

int main() {
    int n, m;
    cin >> n >> m;
    vector<int> mi(n);
    for (int i = 0; i < n; ++i) cin >> mi[i];
    cout << count_solutions_lower_bound(m, mi) << endl;
    return 0;
}
```

(e) **Đề bài:** Đếm số nghiệm nguyên của phương trình:

$$\sum_{i=1}^{n} x_i = m \quad \text{sao cho } m_i \le x_i \le M_i,\ \forall i \in [n]$$

Lời giải: Đặt $y_i = x_i - m_i \Rightarrow 0 \le y_i \le a_i = M_i - m_i$ Khi đó:

$$\sum_{i=1}^{n} y_i = m - \sum_{i=1}^{n} m_i =: T$$

Bài toán trở thành: đếm số nghiệm nguyên không âm y_i sao cho:

$$\sum y_i = T, \quad 0 \le y_i \le a_i$$

Cài đặt C++:

```
// bai_e.cpp
#include <iostream>
#include <vector>
using namespace std;

long long count_bounded_solutions(int m, const vector<int>& mi,
    const vector<int>& Mi) {
    int n = mi.size();
    int T = m;
    for (int x : mi) T -= x;
```

```
    if (T < 0) return 0;

    vector<int> a(n);
    for (int i = 0; i < n; ++i) a[i] = Mi[i] - mi[i];

    vector<long long> dp(T + 1, 0);
    dp[0] = 1;

    for (int i = 0; i < n; ++i) {
        vector<long long> new_dp(T + 1, 0);
        for (int t = 0; t <= T; ++t) {
            for (int k = 0; k <= min(a[i], t); ++k) {
                new_dp[t] += dp[t - k];
            }
        }
        dp = new_dp;
    }
    return dp[T];
}

int main() {
    int n, m;
    cin >> n >> m;
    vector<int> mi(n), Mi(n);
    for (int i = 0; i < n; ++i) cin >> mi[i];
    for (int i = 0; i < n; ++i) cin >> Mi[i];
    cout << count_bounded_solutions(m, mi, Mi) << endl;
    return 0;
}
```

3. Một đồ thị đơn G có 9 cạnh, và bậc của mỗi đỉnh đều ít nhất là 3. Xác định tất cả các khả năng có thể xảy ra cho số đỉnh của đồ thị. Với mỗi khả năng, hãy cho một ví dụ cụ thể.

Tổng bậc của tất cả các đỉnh bằng hai lần số cạnh:

$$\sum_{v \in V} \deg(v) = 2 \cdot |E| = 2 \cdot 9 = 18$$

Gọi số đỉnh là n. Vì mỗi đỉnh có bậc ít nhất là 3, ta có:

$$\sum \deg(v) \geq 3n \Rightarrow 18 \geq 3n \Rightarrow n \leq 6$$

Mặt khác, tổng bậc là 18, nên ta cũng có:

$$3n \leq 18 \Rightarrow n \geq 6$$

Do đó, suy ra:

$$\boxed{n = 6}$$

Vì tổng bậc là 18 và có 6 đỉnh, nên mỗi đỉnh phải có bậc đúng bằng 3:

$$\deg(v_i) = 3 \quad \text{với mọi } i = 1 \dots 6$$

Ví dụ

Xây dựng một đồ thị đều bậc 3 với 6 đỉnh và 9 cạnh:

- Đỉnh: $v_1, v_2, v_3, v_4, v_5, v_6$
- Các cạnh:

$$(v_1, v_2), (v_1, v_3), (v_1, v_4),$$
$$(v_2, v_3), (v_2, v_5), (v_3, v_6),$$
$$(v_4, v_5), (v_4, v_6), (v_5, v_6)$$

Kết luận

Số đỉnh duy nhất thỏa mãn điều kiện là $\boxed{6}$. Đồ thị là một đồ thị đều bậc 3 với 6 đỉnh và 9 cạnh.

4. Đề bài

Xét dãy số:

$$(7, 7, 6, 5, 4, 4, 4, 3, 2)$$

Hỏi: Dãy này có phải là dãy bậc của một đồ thị đơn hay không?

Áp dụng định lý Havel–Hakimi

Ta thực hiện các bước như sau:

Bước 1: Sắp xếp giảm dần: $(7, 7, 6, 5, 4, 4, 4, 3, 2)$
Bỏ số đầu tiên là 7, trừ 1 vào 7 số tiếp theo: $\Rightarrow (6, 5, 4, 3, 3, 3, 2, 2)$

Bước 2: Sắp xếp lại: $(6, 5, 4, 3, 3, 3, 2, 2)$
Bỏ 6, trừ 1 vào 6 số tiếp theo: $\Rightarrow (4, 3, 2, 2, 2, 1, 2)$

Bước 3: Sắp xếp lại: $(4, 3, 2, 2, 2, 2, 1)$
Bỏ 4, trừ 1 vào 4 số tiếp theo: $\Rightarrow (2, 1, 1, 1, 2, 1)$

Bước 4: Sắp xếp lại: $(2, 2, 1, 1, 1, 1)$
Bỏ 2, trừ 1 vào 2 số tiếp theo: $\Rightarrow (1, 0, 1, 1, 1)$

Bước 5: Sắp xếp lại: $(1, 1, 1, 1, 0)$
Bỏ 1, trừ 1 vào 1 số tiếp theo: $\Rightarrow (0, 1, 1, 0)$

Bước 6: Sắp xếp lại: $(1, 1, 0, 0)$
Bỏ 1, trừ 1 vào 1 số tiếp theo: $\Rightarrow (0, 0, 0)$

Bước 7: Dãy toàn số 0 $\Rightarrow$ dãy là dãy đồ thị.

Kết luận

$\boxed{(7, 7, 6, 5, 4, 4, 4, 3, 2) \text{ là dãy đồ thị}}$

5. Giả sử bạn áp dụng thuật toán Havel–Hakimi cho một dãy bậc và đến cuối quá trình, bạn nhận được một dãy đồ thị. Bạn vẽ một đồ thị đơn tương ứng với dãy cuối này và bắt đầu xây dựng ngược lại để tạo thành một đồ thị đơn có dãy bậc ban đầu. Trong quá trình xây dựng ngược, liệu có đúng là tại mỗi bước, khi thêm một đỉnh mới, bạn luôn nối nó với các đỉnh hiện tại có bậc cao nhất? Hãy chứng minh hoặc đưa ra phản ví dụ.

Phân tích thuật toán Havel–Hakimi

Thuật toán Havel–Hakimi kiểm tra một dãy $d = (d_1, d_2, \ldots, d_n)$ có phải là dãy bậc của một đồ thị đơn không. Các bước:

Bước 1: Sắp xếp dãy theo thứ tự giảm dần.

Bước 2: Bỏ phần tử đầu tiên d_1, trừ 1 vào d_1 phần tử tiếp theo.

Bước 3: Nếu xuất hiện số âm hoặc không đủ phần tử để trừ, thì dãy không là đồ thị.

Bước 4: Nếu dãy trở thành toàn 0, thì là dãy đồ thị.

Quá trình xây dựng ngược lại

Khi xây dựng đồ thị ngược từ dãy $(0, 0, \ldots, 0)$ về dãy ban đầu, tại mỗi bước ta thêm một đỉnh mới v với bậc d, và phải nối nó với d đỉnh đã có để tạo ra dãy trước đó.

Một giả thuyết hợp lý là: *ta luôn nối đỉnh mới với các đỉnh có bậc cao nhất hiện tại.* Chúng ta sẽ kiểm tra điều này.

Phản ví dụ

Xét dãy sau:

$$(4, 3, 3, 2, 2, 2)$$

Đây là một dãy đồ thị (có thể kiểm tra bằng Havel–Hakimi).

Giả sử ta xây dựng ngược lại, ta cần thêm các đỉnh từng bước và nối với các đỉnh đã có để khôi phục đúng các bậc. Tuy nhiên, sẽ có bước mà việc nối với các đỉnh có bậc cao nhất không khả thi hoặc gây ra mâu thuẫn (như tạo cạnh trùng lặp hoặc làm một đỉnh vượt quá bậc cho phép).

Một ví dụ đơn giản hơn:

Xét dãy:

$$(3, 3, 3, 3)$$

Là dãy bậc của đồ thị K_4. Khi xây dựng lại:

- Bắt đầu từ $(0, 0) \rightarrow$ thêm đỉnh bậc $1 \rightarrow$ nối với một đỉnh bất kỳ.
- Tiếp theo, thêm đỉnh bậc $2 \rightarrow$ nối với hai đỉnh bất kỳ.
- Cuối cùng thêm đỉnh bậc 3, nhưng không nhất thiết phải nối với ba đỉnh có bậc cao nhất, vì mọi đỉnh đều có cùng bậc.

Như vậy, quá trình nối không bắt buộc phải theo các đỉnh có bậc cao nhất hiện tại.

Kết luận

Không phải lúc nào cũng nối đỉnh mới với các đỉnh có bậc cao nhất hiện tại.

Quá trình xây dựng ngược từ thuật toán Havel–Hakimi không duy nhất, và việc nối phụ thuộc vào khả năng khôi phục đúng bậc, không bắt buộc phải là các đỉnh có bậc lớn nhất.

6. P10.1.7

Giả sử ta có dãy (d_1, d_2, d_3, d_4) với $d_1 \geq d_2 \geq d_3 \geq d_4 \geq 0$ và biết rằng dãy này **không** là dãy đồ thị. Ta xét dãy mới:

$$(d_1, d_2, d_3, d_4 + 1, 1)$$

Hỏi: Dãy mới có thể là dãy đồ thị (sau khi sắp xếp giảm dần) không? Hãy chứng minh hoặc đưa ra phản ví dụ.

Phân tích và phản ví dụ

Xét dãy:

$$(2, 2, 2, 1)$$

Áp dụng thuật toán Havel–Hakimi:

- Bỏ 2 → trừ 1 vào 2 phần tử → $(1, 1, 1)$
- Bỏ 1 → trừ 1 vào 1 phần tử → $(0, 1)$
- Bỏ 1 → trừ 1 vào 1 phần tử → (0) → lỗi (âm)

⇒ Dãy **không** là dãy đồ thị.

Xét dãy mới sau khi thêm $+1$ vào phần tử cuối và thêm 1:

$$(2, 2, 2, 2, 1)$$

Áp dụng Havel–Hakimi:

- Bỏ 2 → $(1, 1, 2, 1)$
- Sắp: $(2, 1, 1, 1)$
- Bỏ 2 → $(0, 0, 1)$
- Sắp: $(1, 0, 0)$
- Bỏ 1 → $(0, 0)$ → OK

⇒ Dãy mới là dãy đồ thị

Kết luận

Một dãy không đồ thị có thể trở thành đồ thị sau khi thêm $+1$ và 1.

7. Đề bài: Prime Path

Cho hai số nguyên tố có 4 chữ số, bạn cần tìm số bước ít nhất để chuyển từ số đầu tiên sang số thứ hai, với các điều kiện sau:

- Mỗi bước chỉ được phép thay đổi đúng **1 chữ số** trong số hiện tại.
- Số mới sau mỗi bước cũng phải là một số nguyên tố có 4 chữ số.

In ra số bước ít nhất. Nếu không thể chuyển đổi, in ra dòng: `Impossible to reach <end> from <start>`.

Ví dụ: `start = 1033, end = 8179`
Kết quả: `Minimum number of steps: 6`

Ý tưởng thuật toán

Ta mô hình hóa các số nguyên tố 4 chữ số như các đỉnh trong đồ thị. Hai đỉnh được nối với nhau nếu chúng khác nhau đúng 1 chữ số và đều là số nguyên tố.

Dùng thuật toán BFS để tìm đường đi ngắn nhất từ số bắt đầu đến số kết thúc.

Mã nguồn C++

```cpp
// Prime Path Solver using BFS

#include <iostream>
#include <queue>
#include <vector>
#include <string>
#include <cstring>

using namespace std;

const int MAX = 10000;
bool isPrime[MAX];

void sieve() {
    memset(isPrime, true, sizeof(isPrime));
    isPrime[0] = isPrime[1] = false;
    for (int i = 2; i * i < MAX; ++i) {
        if (isPrime[i]) {
            for (int j = i * i; j < MAX; j += i)
                isPrime[j] = false;
        }
    }
}

int primePath(int start, int end) {
    queue<pair<int, int>> q;
    bool visited[MAX] = {false};

    q.push(make_pair(start, 0));
    visited[start] = true;

    while (!q.empty()) {
        int curr = q.front().first;
        int steps = q.front().second;
        q.pop();

        if (curr == end) return steps;

        string s = to_string(curr);
        for (int i = 0; i < 4; ++i) {
            char original = s[i];
            for (char d = '0'; d <= '9'; ++d) {
                if (d == original) continue;
                s[i] = d;
                int next = stoi(s);
                if (next >= 1000 && isPrime[next] && !visited[next])
                    {
                    visited[next] = true;
                    q.push(make_pair(next, steps + 1));
```

```
                }
            }
            s[i] = original;
        }
    }
    return -1;
}

int main() {
    sieve();

    int start, end;
    cout << "Nhap so nguyen to bat dau (4 chu so): ";
    cin >> start;
    cout << "Nhap so nguyen to ket thuc (4 chu so): ";
    cin >> end;

    if (!isPrime[start] || !isPrime[end] || start < 1000 || end <
       1000) {
        cout << "Mot hoac ca hai khong phai so nguyen to 4 chu so!"
           << endl;
        return 1;
    }

    int steps = primePath(start, end);

    if (steps == -1)
        cout << "Impossible to reach " << end << " from " << start
           << endl;
    else
        cout << "Minimum number of steps: " << steps << endl;

    return 0;
}
```