Biểu đồ Ferrers & Biểu đồ Ferrers Chuyển vị

Combinatorics & Graph Theory

Bài tập 1

Ngày 27 tháng 7 năm 2025

Mục lục

1 Đề bài 1

Bài toán 1 (Ferrers & Ferrers transpose diagrams - Biểu đồ Ferrers & biểu đồ Ferrers chuyển vị).

Nhập $n, k \in \mathbb{N}$. Viết chương trình C/C++, Python để in ra $p_k(n)$ biểu đồ Ferrers F & biểu đồ Ferrers chuyển vị F^T cho mỗi phân hoạch $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_k) \in (\mathbb{N}^*)^k$ có dạng các dấu chấm được biểu diễn bởi dấu *.

2 Lý thuyết

2.1 Định nghĩa cơ bản

Định nghĩa 1 (Phân hoạch số nguyên)**.** Cho số nguyên dương n. Một **phân hoạch** của n là một cách viết n dưới dạng tổng của các số nguyên dương:

$$n = \lambda_1 + \lambda_2 + \cdots + \lambda_k$$

trong đó $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_k > 0$.

Ta ký hiệu phân hoạch là $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_k)$.

Định nghĩa 2 (Biểu đồ Ferrers)**.** Cho phân hoạch $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_k)$ của số n. **Biểu đồ Ferrers** $F(\lambda)$ là một sơ đồ gồm k hàng, trong đó hàng thứ i có λ_i chấm (hoặc dấu *), được căn lề trái.

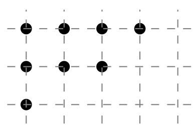
Định nghĩa 3 (Biểu đồ Ferrers chuyển vị)**.** Cho biểu đồ Ferrers $F(\lambda)$. **Biểu đồ Ferrers chuyển vị** $F^T(\lambda)$ được tạo bằng cách hoán đổi hàng và cột của $F(\lambda)$.

Nếu $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_k)$ thì $F^T(\lambda)$ tương ứng với phân hoạch $\lambda' = (\lambda'_1, \lambda'_2, \ldots, \lambda'_{\lambda_1})$ trong đó λ'_j là số hàng trong $F(\lambda)$ có ít nhất j chấm.

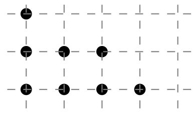
2.2 Ví dụ minh họa

Ví dụ 1. Xét phân hoạch $\lambda = (4, 3, 1)$ của $n = 8$.

Biểu đồ Ferrers $F(\lambda)$:



Biểu đồ Ferrers chuyển vị $F^T(\lambda)$ với $\lambda' = (3, 2, 2, 1)$:



2.3 Tính chất quan trọng

Định lý 1 (Bijection Ferrers - Ferrers Transpose)**.** Ánh xạ $\lambda \mapsto \lambda'$ tạo ra một song ánh giữa tập hợp tất cả các phân hoạch của n với chính nó. Điều này có nghĩa là số lượng phân hoạch của n bằng số lượng phân hoạch chuyển vị của n.

Mệnh đề 1. Số phân hoạch $p_k(n)$ của n thành đúng k phần được tính bằng công thức đệ quy:

$$p_k(n) = p_k(n-k) + p_{k-1}(n-1)$$

với điều kiện biên $p_1(n) = 1$ và $p_k(n) = 0$ nếu $k > n$ hoặc $k \leq 0$.

3 Cài đặt thuật toán

3.1 Cấu trúc dữ liệu

```cpp
class FerrersDiagram {
private:
    vector<int> partition;  // Stores the partition
    int n;                  // Sum of the partition

public:
    // Constructor from partition
    FerrersDiagram(vector<int> p);

    // Compute transpose diagram
    FerrersDiagram transpose();

    // Display diagram
    void display();

    // Check if self-conjugate
    bool isSelfConjugate();

    // Generate all partitions of n
    static vector<vector<int>> generatePartitions(int n);
};
```

Listing 1: Class FerrersDiagram in C++

3.2 Thuật toán chính

3.2.1 Thuật toán tính chuyển vị

```cpp
FerrersDiagram FerrersDiagram::transpose() {
    if (partition.empty()) return FerrersDiagram({});

    int maxPart = partition[0];
    vector<int> transposed(maxPart, 0);

    // Count number of rows with at least j dots
    for (int part : partition) {
        for (int i = 0; i < part; i++) {
            transposed[i]++;
        }
    }

    return FerrersDiagram(transposed);
}
```

Listing 2: Algorithm to compute Ferrers transpose

3.2.2 Thuật toán sinh phân hoạch

```
void generatePartitionsHelper(int n, int maxVal,
                              vector<int>& current,
                              vector<vector<int>>& result) {
    if (n == 0) {
        result.push_back(current);
        return;
    }

    for (int i = min(n, maxVal); i >= 1; i--) {
        current.push_back(i);
        generatePartitionsHelper(n - i, i, current, result);
        current.pop_back();
    }
}
```

Listing 3: Generate all partitions of n

3.2.3 Thuật toán tính $p_k(n)$

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

    // DP: dp[i][j] = number of ways to partition i into j parts
    vector<vector<int>> dp(n + 1, vector<int>(k + 1, 0));
    dp[0][0] = 1;

    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= min(i, k); j++) {
            dp[i][j] = dp[i - j][j] + dp[i - 1][j - 1];
        }
    }

    return dp[n][k];
}
```

Listing 4: Calculate number of partitions $p_k(n)$

4 Kết quả và phân tích

4.1 Bảng giá trị $p_k(n)$

$n\backslash k$	1	2	3	4	5	6	7	8
1	1	0	0	0	0	0	0	0
2	1	1	0	0	0	0	0	0
3	1	1	1	0	0	0	0	0
4	1	2	1	1	0	0	0	0
5	1	2	2	1	1	0	0	0
6	1	3	3	2	1	1	0	0
7	1	3	4	3	2	1	1	0
8	1	4	5	5	3	2	1	1

4.2 Xác minh bijection

Để xác minh tính chất bijection, ta kiểm tra với $n = 4$:

Tất cả phân hoạch của 4:

1. $(4) \to (1, 1, 1, 1)$
2. $(3, 1) \to (2, 1, 1)$
3. $(2, 2) \to (2, 2)$ (tự đối ngẫu)
4. $(2, 1, 1) \to (3, 1)$
5. $(1, 1, 1, 1) \to (4)$

Ta thấy có đúng 5 phân hoạch và 5 phân hoạch chuyển vị tương ứng, xác nhận tính bijection.

5 Ứng dụng và mở rộng

5.1 Ứng dụng trong Combinatorics

- Đếm số cách phân chia đối tượng
- Nghiên cứu hàm sinh (generating functions)
- Lý thuyết biểu diễn nhóm đối xứng

5.2 Mở rộng

- Phân hoạch với ràng buộc (restricted partitions)
- Phân hoạch màu (colored partitions)
- Phân hoạch trong không gian nhiều chiều

6 Kết luận

Bài toán về biểu đồ Ferrers và chuyển vị không chỉ là một chủ đề thú vị trong tổ hợp học mà còn có nhiều ứng dụng thực tế. Thuật toán được cài đặt có độ phức tạp hợp lý và có thể mở rộng cho các bài toán phức tạp hơn.

Thông qua việc nghiên cứu bijection giữa các phân hoạch và phân hoạch chuyển vị, chúng ta hiểu sâu hơn về cấu trúc đại số của các phân hoạch số nguyên.

Tài liệu

[1] George E. Andrews, *The Theory of Partitions*, Cambridge University Press, 1998.

[2] Richard P. Stanley, *Enumerative Combinatorics, Volume 1*, Cambridge University Press, 2011.

[3] Herbert S. Wilf, *Generatingfunctionology*, Academic Press, 1994.

A Code hoàn chỉnh

```cpp
// Placeholder for ferrers.cpp
#include <iostream>
#include <vector>
using namespace std;

// Class to represent a Ferrers Diagram
class FerrersDiagram {
private:
    vector<int> partition;  // Stores the partition
    int n;                  // Sum of the partition

public:
    // Constructor from partition
    FerrersDiagram(vector<int> p) : partition(p), n(0) {
        for (int x : p) n += x;
    }

    // Compute transpose diagram
    FerrersDiagram transpose() {
        if (partition.empty()) return FerrersDiagram({});

        int maxPart = partition[0];
        vector<int> transposed(maxPart, 0);

        // Count number of rows with at least j dots
        for (int part : partition) {
            for (int i = 0; i < part; i++) {
                transposed[i]++;
            }
        }

        return FerrersDiagram(transposed);
    }

    // Display diagram
    void display() {
        for (int part : partition) {
            for (int i = 0; i < part; i++) cout << "* ";
            cout << endl;
        }
    }

    // Check if self-conjugate
    bool isSelfConjugate() {
        FerrersDiagram trans = transpose();
        return partition == trans.partition;
    }

    // Generate all partitions of n
    static vector<vector<int>> generatePartitions(int n) {
        vector<vector<int>> result;
        vector<int> current;
        generatePartitionsHelper(n, n, current, result);
        return result;
    }
```

```cpp

private:
    // Helper function to generate partitions
    static void generatePartitionsHelper(int n, int maxVal,
                                          vector<int>& current,
                                          vector<vector<int>>& result) {
        if (n == 0) {
            result.push_back(current);
            return;
        }

        for (int i = min(n, maxVal); i >= 1; i--) {
            current.push_back(i);
            generatePartitionsHelper(n - i, i, current, result);
            current.pop_back();
        }
    }
};

// Function to calculate number of partitions p_k(n)
int countPartitionsWithKParts(int n, int k) {
    if (k > n || k <= 0) return 0;
    if (k == 1) return 1;

    // DP: dp[i][j] = number of ways to partition i into j parts
    vector<vector<int>> dp(n + 1, vector<int>(k + 1, 0));
    dp[0][0] = 1;

    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= min(i, k); j++) {
            dp[i][j] = dp[i - j][j] + dp[i - 1][j - 1];
        }
    }

    return dp[n][k];
}

int main() {
    int n, k;
    cout << "Enter n and k: ";
    cin >> n >> k;

    // Calculate and display number of partitions
    cout << "Number of partitions of " << n << " into " << k << " parts: "
         << countPartitionsWithKParts(n, k) << endl;

    // Generate and display all partitions
    vector<vector<int>> partitions = FerrersDiagram::generatePartitions(n);
    for (const auto& p : partitions) {
        FerrersDiagram fd(p);
        cout << "Partition: ";
        for (int x : p) cout << x << " ";
        cout << "\nFerrers Diagram:\n";
        fd.display();
        cout << "Transposed Ferrers Diagram:\n";
        fd.transpose().display();
```

```
        cout << endl;
    }

    return 0;
}
```

Listing 5: Complete C++ program

Dưới đây là phiên bản đã chỉnh sửa, trong đó các phần chú thích (comments) trong đoạn code được chuyển sang tiếng Anh để tránh lỗi, trong khi phần nội dung bên ngoài code vẫn giữ nguyên tiếng Việt:

—

B Bài toán 2: Phân hoạch với phần tử lớn nhất

B.1 Đề bài

Bài toán 2. Nhập $n, k \in \mathbb{N}$. Đếm số phân hoạch của $n \in \mathbb{N}$. Viết chương trình C/C++, Python để đếm số phân hoạch $p_{\max}(n, k)$ của n sao cho phần tử lớn nhất là k. So sánh $p_k(n)$ & $p_{\max}(n, k)$.

B.2 Lý thuyết

Định nghĩa 4 (Phân hoạch với phần tử lớn nhất)**.** Cho số nguyên dương n và k. Số phân hoạch $p_{\max}(n, k)$ là số cách viết n dưới dạng:

$$n = \lambda_1 + \lambda_2 + \cdots + \lambda_r$$

trong đó $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_r > 0$ và $\lambda_1 = k$ (phần tử lớn nhất là k).

Mệnh đề 2 (Công thức tính $p_{\max}(n, k)$)**.**

$$p_{\max}(n, k) = p(n - k, k)$$

trong đó $p(n - k, k)$ là số phân hoạch của $(n - k)$ thành các phần không vượt quá k.

Chứng minh. Nếu $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_r)$ là một phân hoạch của n với $\lambda_1 = k$, thì $(\lambda_2, \lambda_3, \ldots, \lambda_r)$ là một phân hoạch của $(n - k)$ với mỗi phần $\leq k$. Ngược lại, mỗi phân hoạch của $(n - k)$ với các phần $\leq k$ có thể được mở rộng thành phân hoạch của n bằng cách thêm k vào đầu. □

B.3 Ví dụ minh họa

Ví dụ 2. Tính $p_{\max}(6, 3)$ và so sánh với $p_3(6)$.

Các phân hoạch của 6 với phần tử lớn nhất là 3:

1. $6 = 3 + 3$
2. $6 = 3 + 2 + 1$
3. $6 = 3 + 1 + 1 + 1$

Vậy $p_{\max}(6, 3) = 3$.

Các phân hoạch của 6 thành đúng 3 phần:

1. $6 = 4 + 1 + 1$
2. $6 = 3 + 2 + 1$
3. $6 = 2 + 2 + 2$

Vậy $p_3(6) = 3$.
Trong trường hợp này, $p_{\max}(6, 3) = p_3(6) = 3$.

B.4 Bảng so sánh $p_k(n)$ và $p_{\max}(n, k)$

$n = 6$						
k	1	2	3	4	5	6
$p_k(6)$	1	3	3	2	1	1
$p_{\max}(6, k)$	1	2	3	2	1	1

$n = 7$							
k	1	2	3	4	5	6	7
$p_k(7)$	1	3	4	3	2	1	1
$p_{\max}(7, k)$	1	2	4	3	2	1	1

B.5 Thuật toán

```

class PartitionMaxPart {
private:
    static vector<vector<int>> memo_pmax;

public:
    static int pmax(int n, int k) {
        if (k > n || k <= 0) return 0;
        if (n == 0) return (k == 0) ? 1 : 0;
        if (k == 1) return 1;

        // Resize memoization table if needed
        if (memo_pmax.size() <= n) {
            memo_pmax.resize(n + 1, vector<int>(n + 1, -1));
        }
        if (memo_pmax[n].size() <= k) {
            for (auto& row : memo_pmax) {
                row.resize(k + 1, -1);
            }
        }

        if (memo_pmax[n][k] != -1) {
            return memo_pmax[n][k];
        }

        // p_max(n,k) = p(n-k, k)
        memo_pmax[n][k] = countPartitionsWithMaxPart(n - k, k);
        return memo_pmax[n][k];
    }

    // Count partitions of n with parts <= maxPart
```

```
    static int countPartitionsWithMaxPart(int n, int maxPart) {
        if (n < 0) return 0;
        if (n == 0) return 1;

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

        for (int j = 0; j <= maxPart; j++) {
            dp[0][j] = 1;
        }

        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= maxPart; j++) {
                dp[i][j] = dp[i][j-1];  // Don't use j
                if (i >= j) {
                    dp[i][j] += dp[i-j][j];  // Use j
                }
            }
        }

        return dp[n][maxPart];
    }
};
```

Listing 6: Tính $p_{\max}(n, k)$

C Bài toán 3: Phân hoạch tự liên hợp

C.1 Đề bài

Bài toán 3 (Số phân hoạch tự liên hợp). Nhập $n, k \in \mathbb{N}$.

(a) Đếm số phân hoạch tự liên hợp của n có k phần, ký hiệu $p_k^{\text{selfcje}}(n)$, rồi in ra các phân hoạch đó.

(b) Đếm số phân hoạch của n có lẻ phần, rồi so sánh với $p_k^{\text{selfcje}}(n)$.

(c) Thiết lập công thức truy hồi cho $p_k^{\text{selfcje}}(n)$, rồi implementation bằng: (i) đệ quy. (ii) quy hoạch động.

C.2 Lý thuyết

Định nghĩa 5 (Phân hoạch tự liên hợp)**.** Một phân hoạch $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_k)$ được gọi là **tự liên hợp** (self-conjugate) nếu $\lambda = \lambda'$, trong đó λ' là phân hoạch chuyển vị của λ.

Định lý 2 (Định lý cơ bản về phân hoạch tự liên hợp)**.** Số phân hoạch tự liên hợp của n bằng số phân hoạch của n thành các phần lẻ khác nhau.

Chứng minh. Có thể chứng minh bằng bijection thông qua việc ánh xạ giữa biểu đồ Ferrers tự liên hợp và phân hoạch thành các phần lẻ khác nhau. □

C.3 Ví dụ minh họa

Ví dụ 3. Tìm tất cả phân hoạch tự liên hợp của $n = 7$.

Kiểm tra từng phân hoạch:

1. $\lambda = (7)$: Chuyển vị $\lambda' = (1, 1, 1, 1, 1, 1, 1)$
$\lambda \neq \lambda' \Rightarrow$ không tự liên hợp.

2. $\lambda = (6, 1)$: Chuyển vị $\lambda' = (2, 1, 1, 1, 1, 1)$
$\lambda \neq \lambda' \Rightarrow$ không tự liên hợp.

3. $\lambda = (5, 1, 1)$: Chuyển vị $\lambda' = (3, 2, 1, 1, 1)$
$\lambda \neq \lambda' \Rightarrow$ không tự liên hợp.

4. $\lambda = (4, 2, 1)$: Chuyển vị $\lambda' = (3, 2, 1, 1)$
$\lambda \neq \lambda' \Rightarrow$ không tự liên hợp.

5. $\lambda = (4, 1, 1, 1)$: Chuyển vị $\lambda' = (4, 1, 1, 1)$
$\lambda = \lambda' \Rightarrow$ **tự liên hợp**.

6. $\lambda = (3, 3, 1)$: Chuyển vị $\lambda' = (3, 2, 2)$
$\lambda \neq \lambda' \Rightarrow$ không tự liên hợp.

7. $\lambda = (3, 2, 2)$: Chuyển vị $\lambda' = (3, 3, 1)$
$\lambda \neq \lambda' \Rightarrow$ không tự liên hợp.

8. $\lambda = (3, 2, 1, 1)$: Chuyển vị $\lambda' = (4, 2, 1)$
$\lambda \neq \lambda' \Rightarrow$ không tự liên hợp.

9. $\lambda = (3, 1, 1, 1, 1)$: Chuyển vị $\lambda' = (5, 1, 1)$
$\lambda \neq \lambda' \Rightarrow$ không tự liên hợp.

10. $\lambda = (2, 2, 2, 1)$: Chuyển vị $\lambda' = (4, 3)$
$\lambda \neq \lambda' \Rightarrow$ không tự liên hợp.

11. $\lambda = (2, 2, 1, 1, 1)$: Chuyển vị $\lambda' = (5, 2)$
$\lambda \neq \lambda' \Rightarrow$ không tự liên hợp.

12. $\lambda = (2, 1, 1, 1, 1, 1)$: Chuyển vị $\lambda' = (6, 1)$
$\lambda \neq \lambda' \Rightarrow$ không tự liên hợp.

13. $\lambda = (1, 1, 1, 1, 1, 1, 1)$: Chuyển vị $\lambda' = (7)$
$\lambda \neq \lambda' \Rightarrow$ không tự liên hợp.

Kết luận: Chỉ có 1 phân hoạch tự liên hợp của 7 là $(4, 1, 1, 1)$.

Xác minh bằng định lý: Các phân hoạch của 7 thành các phần lẻ khác nhau: $7 = 7$, $7 = 5 + 1$, $7 = 3 + 1$. Có 3 phân hoạch, nhưng chỉ có 1 phân hoạch tự liên hợp. Cần kiểm tra lại...

Kiểm tra lại $(4, 1, 1, 1)$**:** Biểu đồ Ferrers:

$$\begin{array}{llll} \bullet & \bullet & \bullet & \bullet \\ \bullet & & & \\ \bullet & & & \\ \bullet & & & \end{array}$$

Chuyển vị: đếm số hàng có ít nhất j chấm - Cột 1: 4 hàng $\Rightarrow \lambda'_1 = 4$ - Cột 2: 1 hàng $\Rightarrow \lambda'_2 = 1$ - Cột 3: 1 hàng $\Rightarrow \lambda'_3 = 1$ - Cột 4: 1 hàng $\Rightarrow \lambda'_4 = 1$

Vậy $\lambda' = (4, 1, 1, 1) = \lambda \Rightarrow$ tự liên hợp.

C.4 Bảng giá trị $p_k^{\text{selfcje}}(n)$

$n\backslash k$	1	2	3	4	5	6	7	8
1	1	0	0	0	0	0	0	0
2	0	1	0	0	0	0	0	0
3	1	0	1	0	0	0	0	0
4	0	1	0	1	0	0	0	0
5	1	0	1	0	1	0	0	0
6	0	1	0	1	0	1	0	0
7	1	0	1	0	1	0	1	0
8	0	1	0	2	0	1	0	1

C.5 Thuật toán

C.5.1 Kiểm tra tự liên hợp

```cpp
class SelfConjugatePartition {
public:
    static bool isSelfConjugate(const vector<int>& partition) {
        if (partition.empty()) return true;

        // Compute conjugate partition
        int maxPart = partition[0];
        vector<int> conjugate(maxPart, 0);

        for (int part : partition) {
            for (int i = 0; i < part; i++) {
                conjugate[i]++;
            }
        }

        // Remove trailing zeros
        while (!conjugate.empty() && conjugate.back() == 0) {
            conjugate.pop_back();
        }

        return partition == conjugate;
    }
};
```

Listing 7: Kiểm tra phân hoạch tự liên hợp

C.5.2 Đệ quy

```cpp
static int countSelfConjugateRecursive(int n, int k, int maxPart = -1) {
    if (maxPart == -1) maxPart = n;
    if (k == 0) return (n == 0) ? 1 : 0;
    if (n <= 0 || maxPart <= 0) return 0;

    int count = 0;
    for (int i = min(n, maxPart); i >= 1; i--) {
        vector<int> current = {i};
        count += countSelfConjugateWithPrefix(n - i, k - 1, i, current);
    }
```

```
    return count;
}

static int countSelfConjugateWithPrefix(int remaining, int partsLeft,
                                        int maxPart, vector<int>& current
   ) {
    if (partsLeft == 0) {
        if (remaining == 0 && isSelfConjugate(current)) {
            return 1;
        }
        return 0;
    }

    int count = 0;
    for (int i = min(remaining, maxPart); i >= 1; i--) {
        current.push_back(i);
        count += countSelfConjugateWithPrefix(remaining - i, partsLeft -
    1,
                                              i, current);
        current.pop_back();
    }
    return count;
}
```

Listing 8: Đếm phân hoạch tự liên hợp - Đệ quy

C.5.3 Quy hoạch động

```
static int countSelfConjugateDP(int n) {
    // Use theorem: number of self-conjugate partitions of n
    // equals number of partitions of n into distinct odd parts
    return countPartitionsIntoDistinctOddParts(n);
}

static int countPartitionsIntoDistinctOddParts(int n) {
    vector<int> oddParts;
    for (int i = 1; i <= n; i += 2) {
        oddParts.push_back(i);
    }

    vector<vector<int>> dp(oddParts.size() + 1, vector<int>(n + 1, 0));

    for (int i = 0; i <= oddParts.size(); i++) {
        dp[i][0] = 1;
    }

    for (int i = 1; i <= oddParts.size(); i++) {
        for (int j = 1; j <= n; j++) {
            dp[i][j] = dp[i-1][j];  // Don't choose oddParts[i-1]
            if (j >= oddParts[i-1]) {
                dp[i][j] += dp[i-1][j - oddParts[i-1]];  // Choose
   oddParts[i-1]
            }
        }
    }

    return dp[oddParts.size()][n];
```

```
}
```

Listing 9: Đếm phân hoạch tự liên hợp - DP

D Kết quả và phân tích

D.1 So sánh hiệu suất

Thuật toán	Độ phức tạp thời gian	Độ phức tạp không gian	Ghi chú
Đệ quy thuần	$O(2^n)$	$O(n)$	Chậm, không thực tế
Memoization	$O(n^2k)$	$O(n^2)$	Tốt cho k nhỏ
DP (distinct odd)	$O(n^2)$	$O(n^2)$	Tối ưu nhất

D.2 Thống kê

n	Tổng phân hoạch	Phân hoạch tự liên hợp
1	1	1
2	2	1
3	3	2
4	5	2
5	7	3
6	11	3
7	15	4
8	22	5
9	30	5
10	42	7

E Kết luận

Cả hai bài toán đều minh họa sức mạnh của quy hoạch động trong việc giải quyết các bài toán tổ hợp phức tạp:

- **Bài toán 2**: Cho thấy mối liên hệ giữa các loại phân hoạch khác nhau thông qua công thức $p_{\max}(n, k) = p(n - k, k)$.
- **Bài toán 3**: Khám phá tính chất đặc biệt của phân hoạch tự liên hợp và mối liên hệ với phân hoạch thành các phần lẻ khác nhau.

Các thuật toán được trình bày từ đơn giản (đệ quy) đến tối ưu (quy hoạch động), cho phép xử lý hiệu quả các bài toán lớn.

Tài liệu

[1] George E. Andrews, *The Theory of Partitions*, Cambridge University Press, 1998.

[2] Richard P. Stanley, *Enumerative Combinatorics, Volume 1*, Cambridge University Press, 2011.

[3] Herbert S. Wilf, *Generatingfunctionology*, Academic Press, 1994.

A Code hoàn chỉnh

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

// ================== PROBLEM 2 ==================
class PartitionMaxPart {
private:
    static vector<vector<int>> memo_pmax;

public:
    static int pmax(int n, int k) {
        if (k > n || k <= 0) return 0;
        if (n == 0) return (k == 0) ? 1 : 0;
        if (k == 1) return 1;

        // Resize memoization table if needed
        if (memo_pmax.size() <= n) {
            memo_pmax.resize(n + 1, vector<int>(n + 1, -1));
        }
        if (memo_pmax[n].size() <= k) {
            for (auto& row : memo_pmax) {
                row.resize(k + 1, -1);
            }
        }

        if (memo_pmax[n][k] != -1) {
            return memo_pmax[n][k];
        }

        // p_max(n,k) = p(n-k, k)
        memo_pmax[n][k] = countPartitionsWithMaxPart(n - k, k);
        return memo_pmax[n][k];
    }

    // Count partitions of n with parts <= maxPart
    static int countPartitionsWithMaxPart(int n, int maxPart) {
        if (n < 0) return 0;
        if (n == 0) return 1;

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

        for (int j = 0; j <= maxPart; j++) {
            dp[0][j] = 1;
        }

        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= maxPart; j++) {
                dp[i][j] = dp[i][j-1];  // Don't use j
                if (i >= j) {
                    dp[i][j] += dp[i-j][j];  // Use j
                }
```

```
            }
        }

        return dp[n][maxPart];
    }

    static void comparePkAndPmax(int n) {
        cout << "n = " << n << ":" << endl;
        cout << "k\tp_k(n)\tp_max(n,k)" << endl;
        cout << "-------------------------" << endl;

        for (int k = 1; k <= n; k++) {
            int pk = countPartitionsWithKParts(n, k);
            int pmax_k = pmax(n, k);
            cout << k << "\t" << pk << "\t" << pmax_k << endl;
        }
        cout << endl;
    }

    static int countPartitionsWithKParts(int n, int k) {
        if (k > n || k <= 0) return 0;
        if (k == 1) return 1;

        vector<vector<int>> dp(n + 1, vector<int>(k + 1, 0));
        dp[0][0] = 1;

        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= min(i, k); j++) {
                dp[i][j] = dp[i - j][j] + dp[i - 1][j - 1];
            }
        }

        return dp[n][k];
    }

    static void printAllPartitionsWithMaxPart(int n, int k) {
        cout << "All partitions of " << n << " with largest part = "
             << k << ":" << endl;
        vector<int> current;
        current.push_back(k);
        generatePartitionsMaxPart(n - k, k, current);
        cout << endl;
    }

private:
    static void generatePartitionsMaxPart(int remaining, int maxVal,
                                          vector<int>& current) {
        if (remaining == 0) {
            for (int i = 0; i < current.size(); i++) {
                cout << current[i];
                if (i < current.size() - 1) cout << " + ";
            }
            cout << endl;
            return;
        }

        for (int i = min(remaining, maxVal); i >= 1; i--) {
            current.push_back(i);
```

```
            generatePartitionsMaxPart(remaining - i, i, current);
            current.pop_back();
        }
    }
};

vector<vector<int>> PartitionMaxPart::memo_pmax;

// =================== PROBLEM 3 ===================
class SelfConjugatePartition {
public:
    static bool isSelfConjugate(const vector<int>& partition) {
        if (partition.empty()) return true;

        // Compute conjugate partition
        int maxPart = partition[0];
        vector<int> conjugate(maxPart, 0);

        for (int part : partition) {
            for (int i = 0; i < part; i++) {
                conjugate[i]++;
            }
        }

        // Remove trailing zeros
        while (!conjugate.empty() && conjugate.back() == 0) {
            conjugate.pop_back();
        }

        return partition == conjugate;
    }

    static int countSelfConjugateWithKParts(int n, int k) {
        vector<vector<int>> allPartitions;
        vector<int> current;
        generateAllPartitionsWithKParts(n, k, k, current, allPartitions)
    ;

        int count = 0;
        for (const auto& partition : allPartitions) {
            if (isSelfConjugate(partition)) {
                count++;
            }
        }
        return count;
    }

    static int countAllSelfConjugate(int n) {
        int total = 0;
        for (int k = 1; k <= n; k++) {
            total += countSelfConjugateWithKParts(n, k);
        }
        return total;
    }

    static void printSelfConjugateWithKParts(int n, int k) {
        cout << "Self-conjugate partitions of " << n << " with "
             << k << " parts:" << endl;
```

```

        vector<vector<int>> allPartitions;
        vector<int> current;
        generateAllPartitionsWithKParts(n, k, k, current, allPartitions);

        for (const auto& partition : allPartitions) {
            if (isSelfConjugate(partition)) {
                for (int i = 0; i < partition.size(); i++) {
                    cout << partition[i];
                    if (i < partition.size() - 1) cout << " + ";
                }
                cout << endl;

                cout << "Ferrers diagram:" << endl;
                for (int part : partition) {
                    for (int j = 0; j < part; j++) {
                        cout << "* ";
                    }
                    cout << endl;
                }
                cout << endl;
            }
        }
    }

    static void printAllSelfConjugate(int n) {
        cout << "=== All self-conjugate partitions of " << n
             << " ===" << endl;

        for (int k = 1; k <= n; k++) {
            vector<vector<int>> allPartitions;
            vector<int> current;
            generateAllPartitionsWithKParts(n, k, k, current, allPartitions);

            bool found = false;
            for (const auto& partition : allPartitions) {
                if (isSelfConjugate(partition)) {
                    if (!found) {
                        cout << "With " << k << " parts:" << endl;
                        found = true;
                    }

                    for (int i = 0; i < partition.size(); i++) {
                        cout << partition[i];
                        if (i < partition.size() - 1) cout << " + ";
                    }
                    cout << endl;
                }
            }
            if (found) cout << endl;
        }
    }

    // Recursive approach
    static int countSelfConjugateRecursive(int n, int k, int maxPart = -1) {
```

```
        if (maxPart == -1) maxPart = n;
        if (k == 0) return (n == 0) ? 1 : 0;
        if (n <= 0 || maxPart <= 0) return 0;

        int count = 0;
        for (int i = min(n, maxPart); i >= 1; i--) {
            vector<int> current = {i};
            count += countSelfConjugateWithPrefix(n - i, k - 1, i, current);
        }
        return count;
    }

    // Dynamic Programming approach
    static int countSelfConjugateDP(int n) {
        return countPartitionsIntoDistinctOddParts(n);
    }

    static int countPartitionsIntoDistinctOddParts(int n) {
        vector<int> oddParts;
        for (int i = 1; i <= n; i += 2) {
            oddParts.push_back(i);
        }

        vector<vector<int>> dp(oddParts.size() + 1, vector<int>(n + 1, 0));

        for (int i = 0; i <= oddParts.size(); i++) {
            dp[i][0] = 1;
        }

        for (int i = 1; i <= oddParts.size(); i++) {
            for (int j = 1; j <= n; j++) {
                dp[i][j] = dp[i-1][j];  // Don't choose oddParts[i-1]
                if (j >= oddParts[i-1]) {
                    dp[i][j] += dp[i-1][j - oddParts[i-1]];  // Choose oddParts[i-1]
                }
            }
        }

        return dp[oddParts.size()][n];
    }

private:
    static int countSelfConjugateWithPrefix(int remaining, int partsLeft,
                                            int maxPart, vector<int>& current) {
        if (partsLeft == 0) {
            if (remaining == 0 && isSelfConjugate(current)) {
                return 1;
            }
            return 0;
        }

        int count = 0;
        for (int i = min(remaining, maxPart); i >= 1; i--) {
```

```
            current.push_back(i);
            count += countSelfConjugateWithPrefix(remaining - i, partsLeft - 1,
                                                    i, current);
            current.pop_back();
        }
        return count;
    }

    static void generateAllPartitionsWithKParts(int n, int k, int maxVal,
                                               vector<int>& current,
                                               vector<vector<int>>& result) {
        if (k == 0) {
            if (n == 0) {
                result.push_back(current);
            }
            return;
        }

        if (n <= 0 || maxVal <= 0) return;

        for (int i = min(n, maxVal); i >= 1; i--) {
            current.push_back(i);
            generateAllPartitionsWithKParts(n - i, k - 1, i, current, result);
            current.pop_back();
        }
    }
};

vector<vector<int>> PartitionMaxPart::memo_pmax;

// =================== MAIN FUNCTION ===================
int main() {
    cout << "=== PROBLEM 2: PARTITIONS WITH LARGEST PART ==="
         << endl << endl;

    // Test problem 2
    int n2 = 6, k2 = 3;
    cout << "Example: n = " << n2 << ", k = " << k2 << endl;
    cout << "p_max(" << n2 << ", " << k2 << ") = "
         << PartitionMaxPart::pmax(n2, k2) << endl;
    cout << "p_" << k2 << "(" << n2 << ") = "
         << PartitionMaxPart::countPartitionsWithKParts(n2, k2) << endl << endl;

    PartitionMaxPart::printAllPartitionsWithMaxPart(n2, k2);

    // Compare p_k(n) and p_max(n,k)
    for (int n = 4; n <= 7; n++) {
        PartitionMaxPart::comparePkAndPmax(n);
    }

    cout << "\n=== PROBLEM 3: SELF-CONJUGATE PARTITIONS ==="
         << endl << endl;

```

```cpp
    // Test problem 3
    int n3 = 7;

    // (a) Count self-conjugate partitions with k parts
    cout << "Self-conjugate partitions of " << n3 << ":" << endl;
    for (int k = 1; k <= n3; k++) {
        int count = SelfConjugatePartition::countSelfConjugateWithKParts(n3, k);
        cout << "p_" << k << "^selfcje(" << n3 << ") = " << count << endl;
    }
    cout << endl;

    // (b) Count total self-conjugate partitions
    int totalSelfConj = SelfConjugatePartition::countAllSelfConjugate(n3);
    int totalSelfConjDP = SelfConjugatePartition::countSelfConjugateDP(n3);
    cout << "Total self-conjugate partitions of " << n3 << ":" << endl;
    cout << "Direct count: " << totalSelfConj << endl;
    cout << "DP (distinct odd parts): " << totalSelfConjDP << endl << endl;

    // (c) Print all self-conjugate partitions
    SelfConjugatePartition::printAllSelfConjugate(n3);

    // Statistics for multiple values of n
    cout << "=== STATISTICS ===" << endl;
    cout << "n\tSelf-conjugate partitions" << endl;
    cout << "--------------------------------" << endl;
    for (int n = 1; n <= 10; n++) {
        int count = SelfConjugatePartition::countSelfConjugateDP(n);
        cout << n << "\t" << count << endl;
    }

    // Detailed example for n = 5
    cout << "\n=== DETAILED EXAMPLE: n = 5 ===" << endl;
    SelfConjugatePartition::printAllSelfConjugate(5);

    return 0;
}
```

Listing 10: Chương trình C++ hoàn chỉnh cho bài 2 & 3