**TUẦN 4: ĐỆ QUY QUAY LUI VÀ KỸ THUẬT NHÁNH CẬN**

Câu hỏi kỳ này:

1. Tìm hiểu về ý tưởng thuật toán đệ quy – quay lui.

2. Hiểu và vận dụng kỹ thuật nhánh cận.

3. Độ phức tạp thuật toán.

**I. GIẢI THUẬT ĐỆ QUY – QUAY LUI (BACKTRACKING).**

1. **Tìm hiểu về đệ quy & giải thuật đệ quy.**
2. **Khái niệm:**

* Đệ quy là**cách lập trình hoặc code một vấn đề, trong đó hàm tự gọi lại chính nó một hoặc nhiều lần trong khối code.**
* Những bài toán giải theo giải thuật đệ quy là những bài toán có thể chia thành những bài toán con nhỏ hơn, đơn giản hơn nhưng vẫn giống với bài toán ban đầu, chia cho đến khi nào được bài toán có thể cho ra được kết quả mà không cần tính toán nữa. Khi đó ta dùng phương pháp quy nạp để tổng hợp lại kết quả, đệ quy chính là cơ sở để hình thành ra giải thuật Chia để Trị mà ta đã học ở tuần 1 và 3.

1. **Tính chất của giải thuật đệ quy.**

* Thủ tục: giải thuật đệ quy gồm 2 phần
  + Phần neo: Bài toán con đơn giản nhất mà ta có thể đọc được kết quả khi không cần tính toán. Ta cũng có thể hiểu đây là phần Trị, thể hiện tính dừng, khi gặp phần neo, ta dừng, không đệ quy tiếp nữa.
  + Phần đệ quy: Khi bài toán con hiện tại chưa đủ đơn giản để giải bằng phần neo, ta vẫn phải tiếp tục đệ quy bài toán hiện tại thành các bài toán con đơn giản hơn nữa, đây chính là phần đệ quy, hay chính là phần Chia. Phần đệ quy thể hiện tính quy nạp.
* Để làm bài theo phương pháp đệ quy, yêu cầu trên hết là ta cần tìm được phần neo, hay chính là điểm dừng đệ quy, để tránh bị đệ quy vô hạn.

1. **Giải thuật đệ quy – quay lui.**
2. **Khái niệm:**

* Quay lui là một giải thuật phát triển thêm từ giải thuật đệ quy. Ý tưởng của quay lui là tìm lời giải từng bước một, mỗi bước chọn một kết quả khả dĩ và thực hiện đệ quy. Bản chất của quay lui chính là liệt kê tất cả các kết quả khả dĩ.
* Ví dụ:

Xét tất cả các giá trị X[1] có thể nhận, thử X[1] nhận các giá trị đó.

Với mỗi giá trị của X[1] ta sẽ: Xét tất cả giá trị X[2] có thể nhận, lại thử X[2] cho các giá trị đó.

Với mỗi giá trị X[2] lại xét khả năng giá trị của X[3]...tiếp tục như vậy cho tới bước: ....

Xét tất cả giá trị X[n] có thể nhận, thử cho X[n] nhận lần lượt giá trị đó.

Thông báo cấu hình tìm được.

* Text

  Description automatically generatedVề bản chất, quay lui chính là đệ quy, nhưng có thêm phần quy lui ở phía sau. Sau khi thực hiện đệ quy, quay lui sẽ được thực hiện để trả về giá trị trước khi đi sâu vào đệ quy.

void Progress(int n)

{

    if(n == 0) return;

    cout << "n = " << n << endl;

    Progress(n - 1);

    cout << "QUAY LUI VE n = " << n << endl;

}

1. **Mô hình thuật toán quay lui.**

void Progress(int i)

{

    for (mọi giá trị j có thể gán cho x[i])

{

        <thử cho x[i] = j>;

        <đánh dấu đã chọn x[i] nếu cần>;

        if (x[i] là phần tử cuối của cấu hình)

{

            <thông báo cấu hình tìm được>;

        }

        else Try(i+1);//gọi đệ quy để chọn tiếp x[i+1]

        <xóa đánh dấu đã chọn x[i] nếu cần>;

    }

}

1. Nhận xét:

* Ưu điểm: Quay lui là thử tất cả các trường hợp để tìm ra lời giải. Thế mạnh là có thể tránh được nhiều trường hợp thiếu hoàn chỉnh, nhờ đó, giảm thời gian chạy.
* Nhược điểm: Trường hợp xấu nhất, độ phức tạp của giải thuật quay lui vẫn là cấp số mũ, cho nên chỉ được áp dụng nhiều ở các bài toán có ít trường hợp.

**II. KỸ THUẬT NHÁNH CẬN (BRANCH AND BOUND).**

1. **Đặt vấn đề.**

* Như đã đề cập ở trên, quay lui chính là liệt kê tất cả các tổ hợp kết quả khả dĩ. Đối với những tổ hợp nhiều phần tử, số lượng tổ hợp sinh ra là vô cùng lớn. Ví dụ khi sinh nhị phân dãy có độ dài n = 30, số tổ hợp cần sinh ra là khoảng hơn 1 tỉ. Thời gian để máy tính hiện nay thực hiện các bài toán tương tự là vô cùng lớn, ta thấy đây là việc làm thiếu tối ưu.
* Khi đó, để tối ưu phương pháp quay lui, ta cần dựa vào những kết quả đã và đang có, để loại đi những phương án mà tương lai chắc chắn không tối ưu. Đó chính là kỹ thuật đánh giá nhánh cận trong tiến trình quay lui, hay ngắn gọn là nhánh cận.

1. **Mô hình giải thuật nhánh cận.**

void branch\_and\_bound(i)

{

    // Đánh giá các nghiệm mở rộng

    if ({Các\_nghiệm\_mở\_rộng\_đều\_không\_tốt\_hơn\_best\_solution})

        return;

    for (value in S[i])

    {

        x[i] = value; // Ghi nhận thành phần thứ i.

        if ({Tìm\_thấy\_nghiệm})

            best\_solution = X; // Cập nhật lại best\_solution bằng nghiệm vừa tìm được.

        else

            branch\_and\_bound(i + 1); // Chưa tìm thấy nghiệm thì xây dựng tiếp.

        {Loại\_bỏ\_thành\_phần\_thứ\_i}

    }

}

* Giải thích: khác với Quay lui, Nhánh cận có thêm bước so sánh giá trị đang xét với giá trị Best\_solution, nếu không tốt hơn, thì ngay lập tức bỏ qua giá trị hiện tại mà không cần đi tới bước đệ quy quay lui, điều đó làm giảm đi đáng kể thời gian chạy code. Nếu giá trị đang xét dẫn tới giá trị tốt hơn Best\_solution hiện tại, thì Best\_solution sẽ được cập nhật.
* Khi ta đánh giá Nhánh cận càng chặt, thì kết quả càng được tối ưu. Tuy nhiên, việc đánh giá hiệu quả lại không hề dễ dàng và dễ gặp sai sót.

**III. BÀI TOÁN THAM KHẢO.**

1. **Bài toán xếp quân hậu.**

Link bài toán: [Sắp xếp quân hậu - QUEEN SPOJ](https://www.spoj.com/PTIT/problems/BCQUEEN/)

Cho một bàn cờ vua có kích thước n \* n, ta biết ràng quân hậu có thể di chuyển theo chiều ngang, dọc, chéo. Vấn đề đặt ra rằng, có n quân hậu, bạn cần đếm số cách đặt n quân hậu này lên bàn cờ sao cho với 2 quân hậu bất kì, chúng không “ăn” nhau.

Input

Một số nguyên n duy nhất (n <= 10)

Output

Số cách đặt quân hậu.

Example

**Input:**

4

**Output:**

2

Ý tưởng:

Ta xét lần lượt các ô trong cột đầu tiên, mỗi ô đó, ta sẽ kiểm tra trong hàng đó đã có chứa quân hậu hâu hay chưa.

Sau đó, ta kiểm tra 2 đường chéo đi qua điểm [i][j] đang xét.

Nếu thỏa mãn yêu cầu, ta đặt 1 quân hậu ở đó, rồi tiếp tục xét đến quân hậu tiếp theo. Ngược lại, nếu không thỏa mãn, ta trả về giá trị của ô trống, rồi xét đến ô kế tiếp nó.

Đến khi nào đạt được hàng cuối cùng, ta tăng biến đếm lên 1 và thử vị trí đầu tiên tiếp theo.

Code mẫu

#include <bits/stdc++.h>

using namespace std;

#define endl "\n"

#define ll long long

const ll big = 1e6;

        // a là cột đi qua [i][j]

        // c1 là đường chéo chính đi qua điểm [i][j]

        // c2 là đường chéo phụ đi qua điểm [i][j]

int n, a[150], c1[150], c2[150], cnt;

void KhoiTao(int n)

{

    for(int i = 0; i < 150; i++)

    {

        a[i] = c1[i] = c2[i] = 0;

    }

}

void Progress(int hang)

{

    for(int i = 0; i < n; i++)

    {

        if(a[i] + c1[hang + i] + c2[hang - i + n - 1] == 0)

        {

            a[i] = c1[hang + i] = c2[hang - i + n - 1] = 1;

            if(hang == n - 1) cnt++;

            Progress(hang + 1);

            a[i] = c1[hang + i] = c2[hang - i + n - 1] = 0;

        }

    }

}

int main()

{

    ios\_base::sync\_with\_stdio(false);cin.tie(0);cout.tie(0);

    int t;

    cin >> t;

    while(t--)

    {

        cin >> n;

        cnt = 0;

        KhoiTao(n);

        Progress(0);

        cout << cnt << endl;

    }

}

1. Bài toán Xâu đặc biệt.

Link bài toán: [Xâu đặc biệt - P135SUMG SPOJ](https://www.spoj.com/PTIT/problems/P135SUMG/)

Trong giờ học, Tí đã viết xong đáp án cho bài tập của mình trên bảng, nhưng trong giờ ra chơi, Tèo lại chọc người bạn của mình bằng cách xóa đi một vài kí tự và thay vào đó bằng các kí tự ‘\_’.

Sau đó, cô giáo đã phát hiện ra trò nghịch của Tèo và ra một hình phạt cho Tèo. Tuy nhiên, Tèo vẫn có thể thoát tội nếu như hoàn thành được bài tập khó của cô giáo. Đó là đếm số lượng các xâu đặc biệt từ xâu mà Tèo vừa sửa của Tí bằng cách thay các kí tự ‘\_’ bằng các kí tự chữ cái in hoa trong bảng chữ cái tiếng anh.

Cô giáo coi một xâu đặc biệt là xâu không chứa 3 nguyên âm liên tiếp, không chứa 3 phụ âm liên tiếp và có chứa ít nhất một kí tự ‘L’.

Input

Một dòng duy nhất là một xâu kí tự có độ dài lớn nhất bằng 100. Xâu này chỉ chứa các chữ cái in hoa và kí tự ‘\_’. Có nhiều nhất 10 kí tự ‘\_’.

Trong bảng chữ cái tiếng anh có 5 nguyên âm là A, E, I, O, U. Các chữ cái còn lại là phụ âm.

Output

In ra một số nguyên duy nhất là số lượng các xâu đặc biệt có thể tạo ra.

Example

**Test 1:**

**Input:**

L\_V

**Output:**

5

**Test 2:**

**Input:**

V\_\_K

**Output:**

10