**Đối với nhiều thuật toán nguyên lý “chia để trị” thường đóng vai trò chủ đạo trong việc thiết kế thuật toán: để giải quyết một bài toán lớn, chúng ta chia nó thành nhiều bài toán con có thể được giải quyết độc lập. Trong phương pháp quy hoạch động, việc thể hiện nguyên lý này được đẩy đến cực độ: khi không biết chắc cần giải quyết bài toán con nào, chúng ta giải quyết tất cả những bài toán con và lưu trữ những lời giải này với mục đích sử dụng lại chúng theo một sự phối hợp nào đó để giải quyết các bài toán tổng quát hơn.**

Thuật ngữ “ quy hoạch” ở đây ngụ ý nói đến quá trình đưa bài toán ban đầu về một dạng nào đó có thể áp dụng phương pháp này để giải. Đó là cả một nghệ thuật, do vậy, bài viết này chỉ xem xét một vài ví dụ nhỏ để qua đó giới thiệu cùng bạn đọc cách tiếp cận này.

Khi sử dụng phương pháp quy hoạch động để giải quyết vấn đề, ta có thể gặp phải hai khó khăn sau:

- Một là, không phải lúc nào sự kết hợp lời giải của bài toán con cũng cho ra lời giải bài toán lớn hơn

- Hai là, số lượng các bài toán con cần giải quyết và lưu trữ đáp án có thể rất lớn, không thể chấp nhận được. Cho đến nay, chưa ai xác định được một cách chính xác những bài nào có thể được giải quyết hiệu quả bằng phương pháp quy hoạch động. Có những vấn đề quá phức tạp và khó khăn mà xem ra không thể ứng dụng quy hoạch động để giải quyết được, trong khi cũng có những bài toán quá đơn giản khiến cho việc sử dụng quy hoạch động để giải quyết lại kém hiệu quả hơn so với dùng các thuạt toán kinh điển.

Chúng ta sẽ cùng xem xét một vài ví dụ cụ thể sử dụng quy hoạch động dưới đây:

**Ví dụ 1: Cho một xâu S <= 1000 kí tự; tìm palindrome dài nhất là xâu con của S ( Xâu con là một dãy các kí tự liên tiếp ).**

(*Palindrome hay còn gọi là xâu đối xứng, xâu đối xứng là tên gọi của những xâu kí tự mà khi viết từ phải qua trái hay từ trái qua phải thì xâu đó không thay đổi. VD: MADAM, IOI,...* )  
**Cách 1: Dùng phương pháp quy hoạch động**

Dùng mảng F[i, j] có ý nghĩa: F[i, j] = true/false nếu đoạn gồm các kí tự từ i đến j của S có/không là palindrome.  
Ta có công thức là:  
\* F[i, i] = True  
\* F[i, j] = F[i+1, j-1]; ( nếu s[ i ] = s[ j ] )  
\* F[i, j] = False; ( nếu s[ i ] <> s[j] )  
Đoạn chương trình như sau:

FillChar( F, sizeof(F), false );

**for** i := 1 **to**n **do** F[i, i] := True;

**for** k := 1 **to** (n-1) **do**

**for** i := 1 **to** (n-k) **do**

**begin**

j := i + k;

F[i, j] := ( F[i+1, j-1] ) **and** (s[ i ] = s[ j ] );

**end**;

Kết quả là : Max(j-i+1) <=j thỏa F[i,j] = True.  
Với phương pháp này, ta thấy độ phức tạp thuật toán là 0(N2).  
***Lưu ý*** : Với N lớn, ta phải thay mảng 2 chiều F bằng 3 mảng 1 chiều và dùng thêm biến max lưu giá trị tối ưu.  
  
**Cách 2: Phương pháp duyệt có cận.**  
Ta xét từng vị trí i:  
+ xem a[ i ] có phải là tâm của Palindrome có lẻ kí tự không?  
( ví dụ Palindrome MADAM có tâm là kí tự D )  
+ xem a[ i ] và a[ i+1 ] có phải là tâm của Palindrome có chẵn kí tự không?  
( ví dụ Palindrome ABBA có tâm là 2 kí tự BB )  
Với mỗi kí tự ta tìm palindrome dài nhất nhận nó là tâm, cập nhập lại kết quả khi duyệt. Ta duyệt từ giữa ra để dùng kết quả hiện tại làm cận.  
Và ta có chương trình sau:

**procedure** Pacondainhat;

**var** i, j : Longint ;

**procedure** Pacon( first, last : Longint );

**var** dd : Longint;

**begin**

**if** first = last **then**

**begin** dd := 1; dec(first); inc(last); **end**

**else** dd := 0;

**repeat**

**if** (first < 1) or (last > N) **then** break;

**if** s[ i ] = s[ j ] **then**

**begin**

dd := dd + 2;

first := first - 1;

last := last + 1;

**end**

**else** break;

**until** false;

**if** max < dd **then** max := dd;

**end**;

{ }

**begin**

i := n **div** 2;

j := n **div** 2 + 1;

**max** := 1;

**while** (i > max **div** 2) **and** (j <= N-max **div** 2) **do**

**begin**

**if** i > max **div** 2 **then**

**begin**

try( i, i );

try( i, i+1 );

**end**;

**if** j <= N - max **div** 2 **then**

**begin**

try( j, j );

try( j, j+1 );

**end**;

i := i - 1;

j := j + 1;

**end**;

**end**;

Cách làm này có độ phức tạp: max\*(N-max). Vì vậy nó chạy nhanh hơn cách quy hoạch động (QHĐ) trên, thời gian chậm nhất khi max = N/2 cũng chỉ mất N2/4 nhanh gấp 4 lần cách dùng QHĐ. Nhờ vậy, chúng ta biết là: không phải lúc nào QHĐ cũng chấp nhận được về mặt thời gian và không phải lúc nào duyệt lúc nào cũng chậm.

**Ví dụ 2: Chia một xâu thành ít nhất các Palindrome (độ dài <=1000). Ví dụ này phức tạp hơn vídụ1, cách làm chúng ta vẫn sử dụng QHĐ.**

Gọi F[ i ] là số palindrome ít nhất mà đoạn 1..j chia thành được.  
Ta có công thức:  
F[ i ] = max( F[ j ] + 1; "j < i thỏa mãn:đoạn j+1..i là palindrome)  
Ta có đoạn chương trình sau:

F[0] := 0;

**for** i := 1 **to** n **do**

**begin**

**for** j := i-1 **downto** 0 **do**

**if** (đoạn j+1..i là palindrome) **then** F[ i ] := max( F[ i ], F[ j ]+1 );

**end**;

Hai vòng **for** lồng nhau mất O(N2), phần kiểm tra đoạn j+1..i là palindrome hay không mất O(N), vậy độ phức tạp thuật toán là O(N3). Sẽ không được khả thi nếu N = 1000. Để giảm độ phức tạp thuật toán, ta sử dụng mảng L[i, j] có ý nghĩa tương tự như mảng F[i, j] ở Vídụ1. QHĐ lập mảng L[i, j] mất N2. Tổng cộng là O(N2) vì mỗi lần kiểm tra chỉ mất O(1).  
Nhưng đến đây lại nảy sinh vấn đề: mảng L[i, j] không thể lưu được khi N=1000 vì bộ nhớ của chúng ta có hạn. Một cách khắc phục là dùng xử lý bít. Nhưng có cách đơn giản hơn là dùng hai mảng một chiều L[ i ] và C[ i ] có ý nghĩa:  
\* L[ i ] là độ dài lớn nhất của palindrome độ dài lẻ nhận s[ i ] làm tâm;  
\* C[ i ] là độ dài lớn nhất của palindrome độ dài chẵn nhận s[ i ] và s[ i+1 ] làm tâm;  
L[ i ] và C[ i ] có thể tính được bằng cách 2 bài 2 trong O(N2). Phần kiểm tra ta viết lại như sau:

**function** is\_palindrome(i, j : integer) : **boolean**;

**var** dd : **integer**;

**begin**

dd := j-i+1;

**if**  dd then is\_palindrome := (L[(i+j) div 2] >= n)

**else** is\_palindrome := (C[(i+j) div 2] >= n)

**end**;

Vậy thuật toán của chúng ta có độ phức tạp tính toán là O(N2), chi phí bộ nhớ là O(N).

**Ví dụ 3 : Cho một xâu, hỏi nó có bao nhiêu xâu con là palindrome; xâu con ở đây gồm các kí tự không cần liên tiếp ( độ dài <= 120 ).**

Ví Dụ: với xâu IOICAMP ta tìm thấy 9 xâu con.  
Đây là một bài tập rất thú vị đơn giản nhưng ứng dụng nhiều trong thực tế. Phương pháp là dùng quy hoạch động.  
Gọi F[i, j] là số palindrome là xâu con của đoạn i..j.  
Ta có công thức :  
\* F[i, i] = 1;  
\* F[i, j] = F[i+1, j] + F[i, j-1] - F[i+1, j-1] + T;  
Nếu s[ i ] = s[j] thì T = F[i+1, j-1] + 1;  
Nếu s[ i ] <> s[j] thì T = 0;  
Đoạn chương trình như sau :

**procedure** Pacon;

**var** k, i, j : **integer**;

**begin**

n := length(s);

**for** i := 1 **to** n **do** F[i, i] := 1;

**for** k := 1 **to** n **do**

**for** i := 1 **to** n-k **do**

**begin**

j := i+k;

F[i, j] := F[i, j-1] + F[i+1, j];

**if** s[ i ] = s[ j ] **then** F[i, j] := F[i, j] + 1

**else** F[i, j] := F[i, j] - F[i+1, j-1];

**end**;

**end**;

Đoạn chương trình trên chỉ có tính mô phỏng, muốn hoàn thiện bạn phải cài đặt các phép tính cộng trừ số lớn vì kết quả có thể lên tới 2n-1.Độ phức tạp của thuật toán là O(N2). Vì vậy, chúng ta hoàn toàn có thể làm với N = 1000, khí đó cần rút gọn mảng F thành ba mảng một chiều.