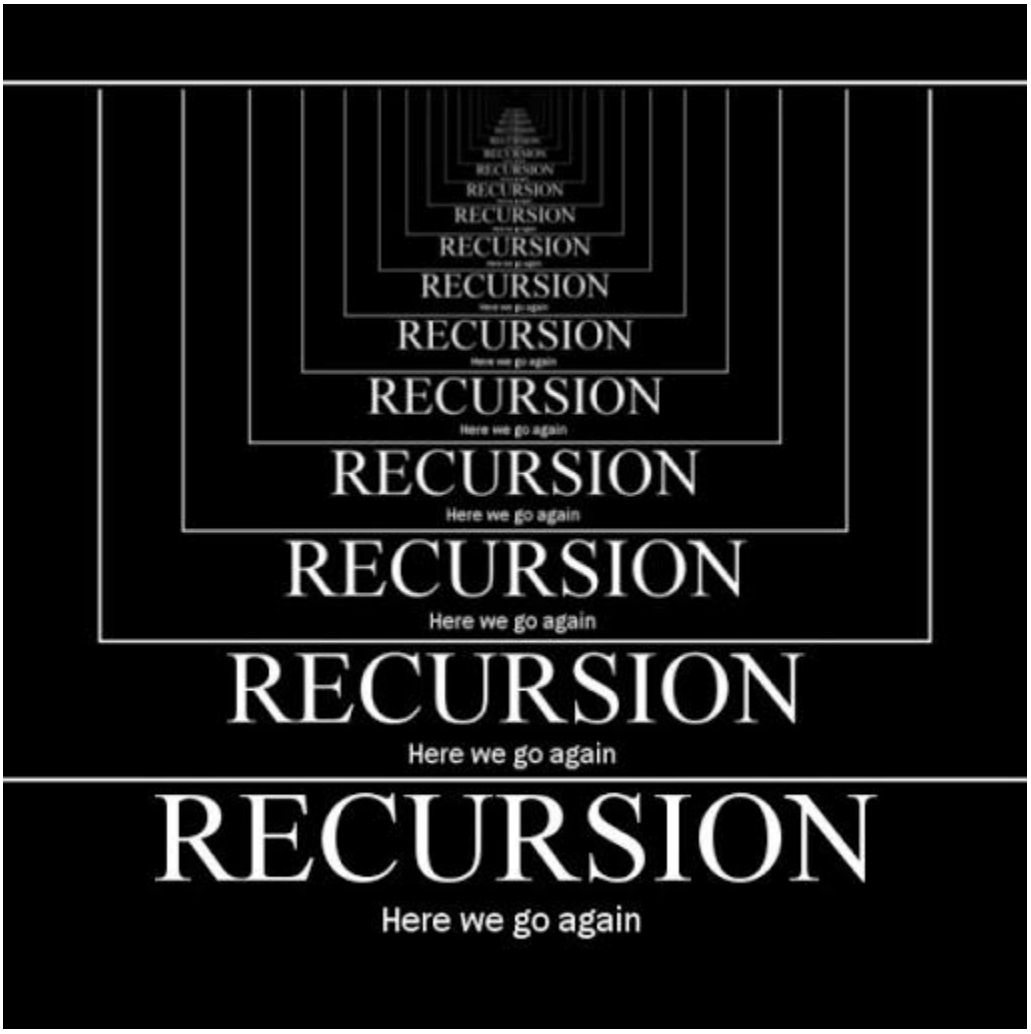
Đệ quy

(cơ bản)

Nguyễn Trung Thành

[2016-02-27]



Mục lục

1. Các vấn đề mở đầu

1a. Bài toán lớn bắt đầu từ bài toán nhỏ

Bài toán: tính tổng S(n) = 1 + 2 + 3 + … + n (n ∈ N*).

Giả sử ta cần tính S(4)

S(4) = 1 + 2 + 3 + 4 = (1 + 2 + 3) + 4 = S(3) + 4

⇨ Để tính được S(4), ta tính S(3)

S(3) = 1 + 2 + 3 = (1 + 2) + 3 = S(2) + 3

⇨ Để tính được S(3), ta tính S(2)

S(2) = 1 + 2 = S(1) + 2

⇨ Để tính được S(2), ta tính S(1)

S(1) = 1

Rõ ràng S(1) = 1 là bài toán cơ bản nhất, ta không cần quy về tính bài toán nào nữa.

Kết luận: để tính S(4) => ta tính S(3) => ta tính S(2) => ta tính S(1) = 1.

Bài toán tính S(4) đã quy về bài toán đơn giản nhất đó là tính S(1) = 1.

1b. Định nghĩa cơ bản

Định nghĩa số tự nhiên:
- 0 là số tự nhiên.
- n là số tự nhiên nếu n-1 là số tự nhiên.

Giả sử ta cần chứng minh 3 là số tự nhiên theo định nghĩa trên.

3 là số tự nhiên nếu 2 là số tự nhiên.
2 là số tự nhiên nếu 1 là số tự nhiên.
1 là số tự nhiên nếu 0 là số tự nhiên.
Mà 0 là số tự nhiên.
Suy ra 3 là số tự nhiên (khá giống logic Toán Rời rạc).

Các ví dụ trên đều cho thấy một ý niệm nào đó đơn giản nhất về đệ quy: bài toán lớn bắt đầu từ bài toán nhỏ hơn, đến khi gặp "bài toán mini" có lời giải sẵn thì xem như ta đã giải được bài toán lớn.

Suy nghĩ :

BT1. Hãy suy nghĩ, phác họa theo sơ đồ cách tính n! theo 2 cách :

+ Dùng vòng lặp for.
+ Dùng đệ quy.

BT2. Hãy suy nghĩ lại bài toán tính S(n) = 1 + 2 + 3 + … + n theo 3 cách :

+ Dùng vòng lặp for.
+ Dùng đệ quy.
+ Dùng công thức tường minh đã học ở lớp 9.

2. Định nghĩa đệ quy

2a. Định nghĩa

Đệ quy là định nghĩa một khái niệm dựa trên chính nó, một hàm gọi đến chính hàm đó.

Ví dụ:

S(n) = 1 + 2 + 3 + … + n = S(n-1) + n.

⇨ S(n) được định nghĩa bởi S(n-1).

⇨ Hàm S tự gọi chính nó.

2b. Tính chất

- **Tính dừng (rất quan trọng) :**
  Đệ quy là phải có điểm dừng, điểm dừng có thể tạm thời hiểu là bài toán đơn giản nhất.
  Ví dụ với S(n) = 1 + 2 + 3 + … + n thì điểm dừng là bài toán đơn giản nhất:
  S(1) = 1.

- Tính định nghĩa :
  Đệ quy mang tính định nghĩa cao, dễ hiểu được bản chất của vấn đề.

- Ưu điểm của đệ quy :
  + Dễ hiểu được bản chất của bài toán.
  + Đa số code đều ngắn.

- Khuyết điểm của đệ quy :
  + Khó kiểm soát được từng bước đi khi lặp lại đệ quy.
  + Bùng nổ tổ hợp: hiểu nôm na là việc một bài toán lớn bị phân thành rất nhiều bài toán nhỏ, dẫn đến gọi hàng ngàn, hàng triệu lần hàm đệ quy và cuối cùng là…treo máy.

Vì vậy, khi giải một bài toán bằng kỹ thuật đệ quy, phải hết sức thận trọng, nếu không mình sẽ không kiểm soát nổi, không cách nào debug nổi, thậm chí treo máy.

Lưu ý khi sử dụng đệ quy trong lập trình thì hay đề cập đến công thức truy hồi hơn là công thức tường minh.

Ví dụ: tính S(n) = 1 + 2 + 3 + … + n.

Trong lập trình, ta hay nghĩ đến công thức truy hồi như sau :

$$\begin{cases} S(1) = 1 \\ S(n) = S(n-1) + n \end{cases}$$

Ta ít khi nghĩ đến công thức tường minh là $S(n) = \frac{n * (n+1)}{2}$

Đối với những bài toán đơn giản, sử dụng công thức tường minh là tốt, nhanh nhưng không còn mang bản chất đệ quy nữa. Khi gặp những bài toán phức tạp, công thức tường minh không thể tìm được, vô dụng :).

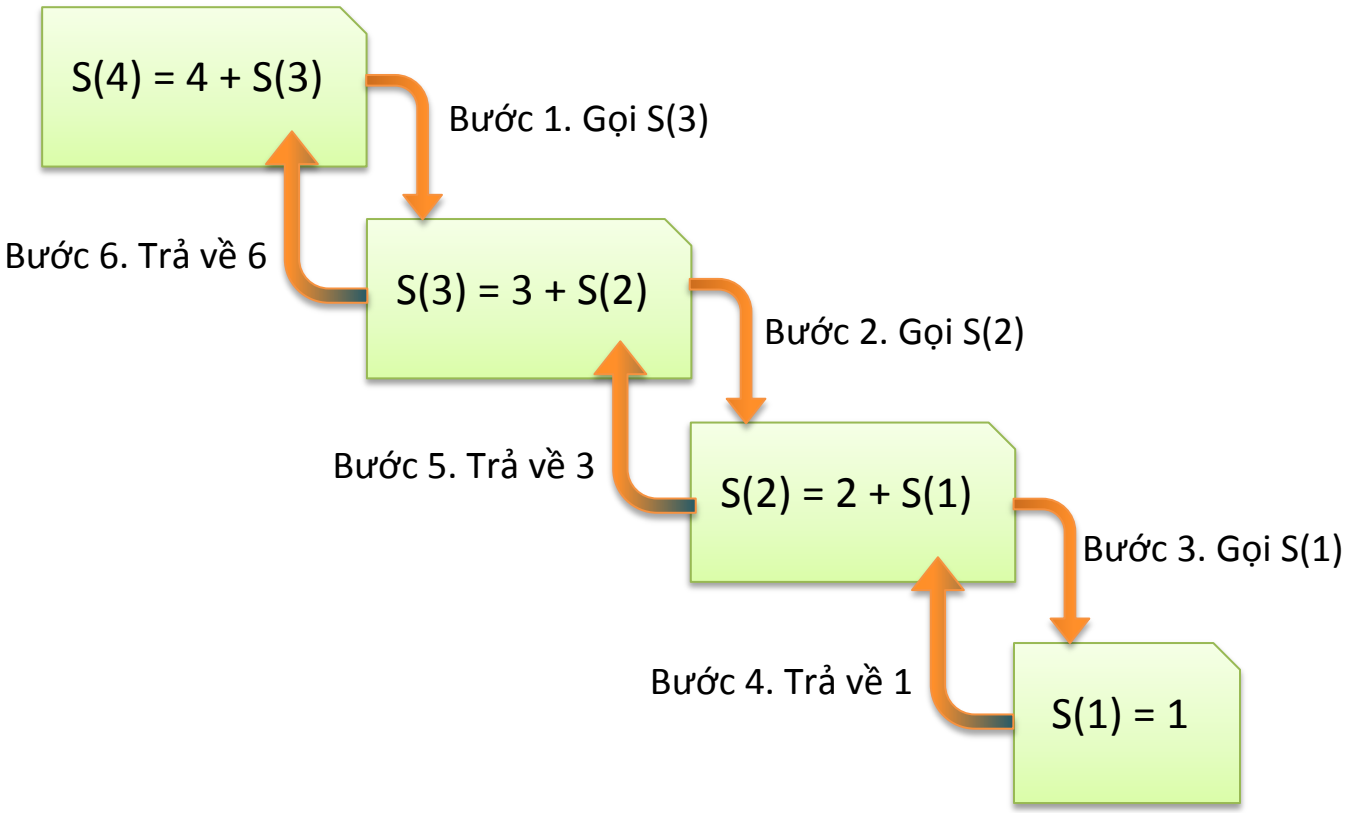
2c. Các ví dụ minh họa

Ví dụ 1 : tính S(n) = 1 + 2 + 3 + … + n (n ≥ 1).

Công thức truy hồi :

$$\begin{cases} S(1) = 1 \\ S(n) = S(n-1) + n \end{cases}$$

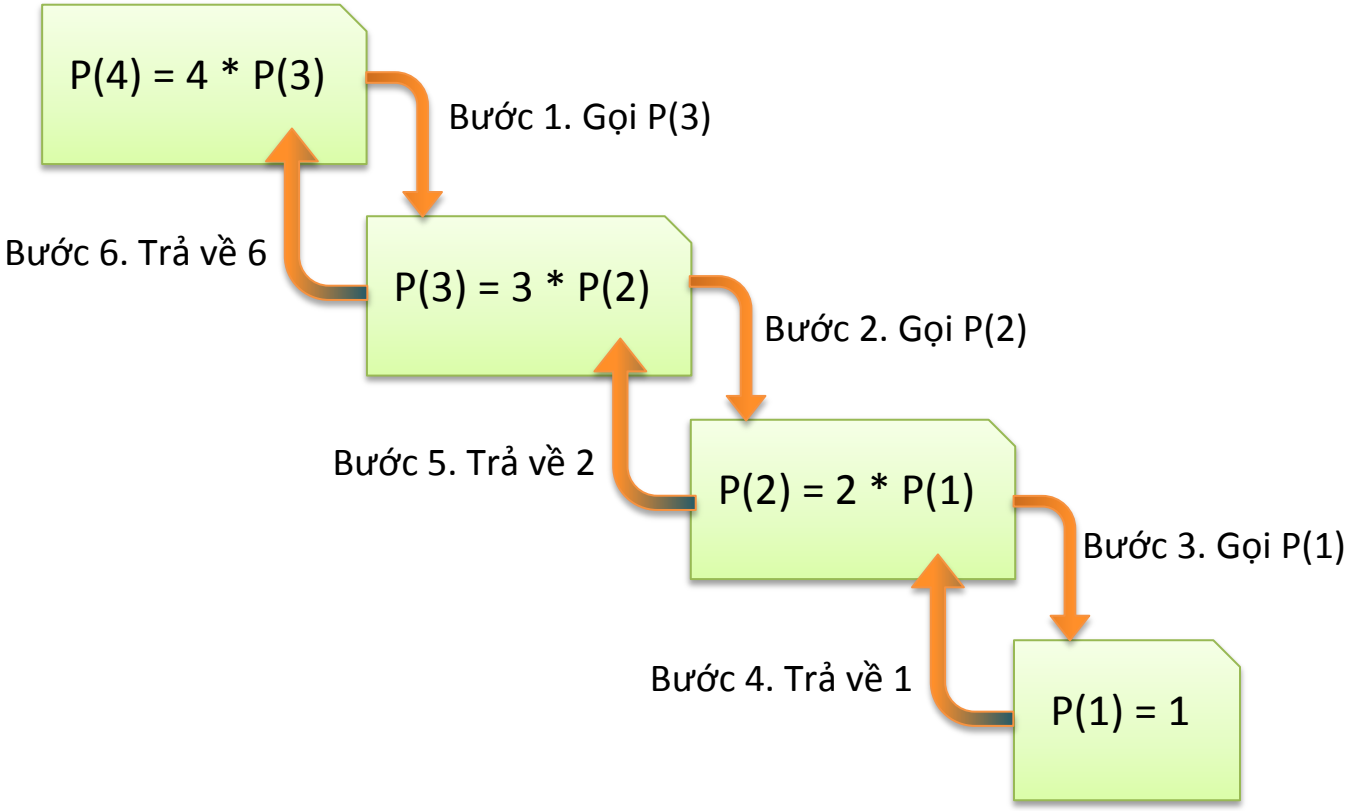
Giả sử ta cần tính S(4).



Và cuối cùng là bước 7: S(4) = 4 + 6 = 10.

Ví dụ 2 : tính P(n) = 1 * 2 * 3 * … * n (n ≥ 1).

Giả sử ta cần tính P(4).



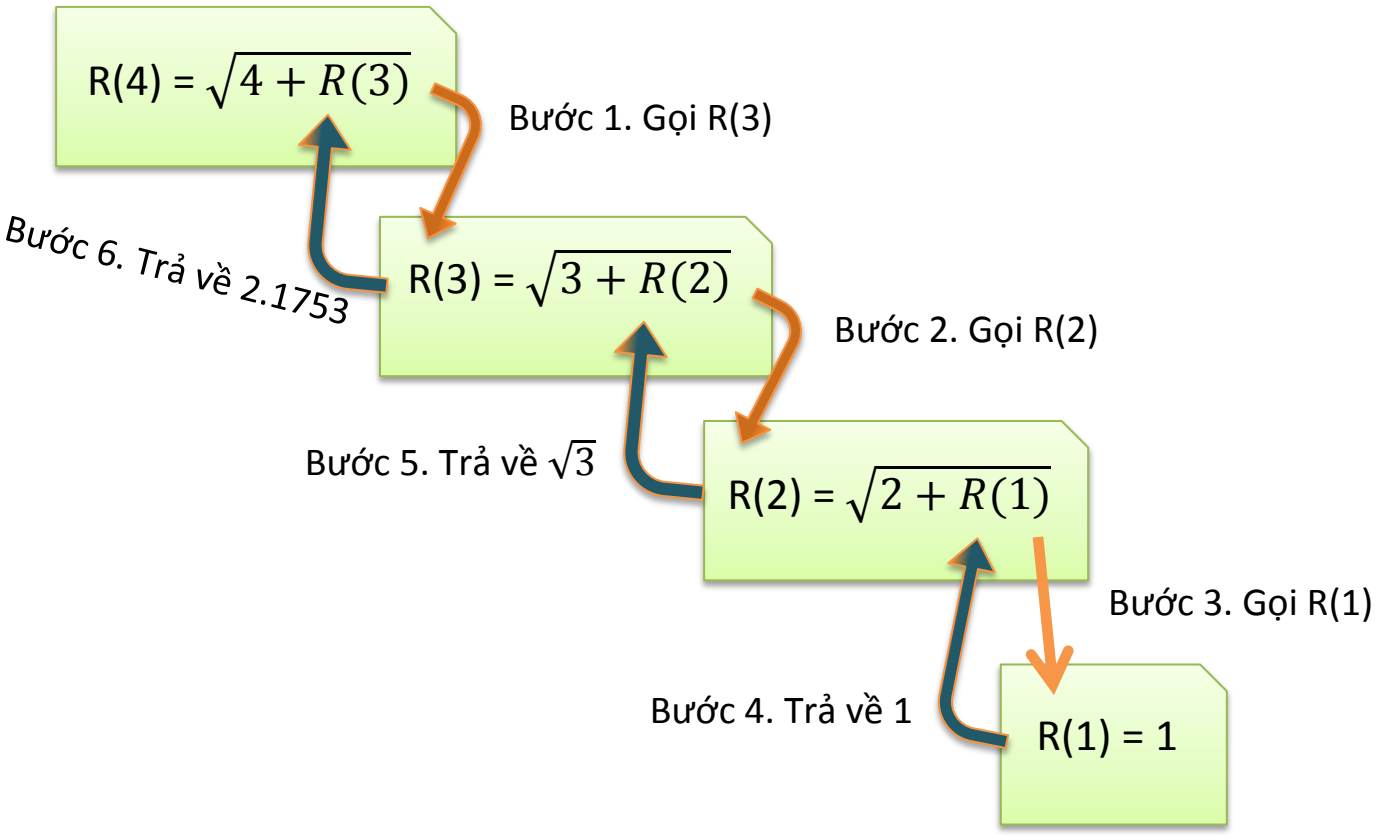
Và cuối cùng là bước 7: P(4) = 4 * 6 = 24.

Ví dụ 3 : tính R(n) = $\sqrt{n+\sqrt{(n-1)+\sqrt{(n-2)+\sqrt{\ldots+\sqrt{1}}}}}$ (n dấu căn).

Giả sử ta cần tính R(4) = $\sqrt{4+\sqrt{3+\sqrt{2+\sqrt{1}}}}$.

Phân tích: Ta có R(2) = $\sqrt{2+\sqrt{1}}$ và R(1) = $\sqrt{1}$.

⇨ R(n) = $\sqrt{n+R(n-1)}$. Kiểm tra lại thì ta thấy công thức đúng.



Và cuối cùng là bước 7: R(4) = $\sqrt{4+2.1753} \approx 2.485$

Bắt đầu khó hiểu

3. Luyện tập viết code

3a. Trình tự phân tích một bài toán theo kiểu đệ quy đơn giản nhất

Bước 1. Viết S(n) và S(n-1).

Bước 2. Tìm mối liên hệ giữa S(n) và S(n-1). Biểu diễn S(n) theo S(n-1).

Bước 3. Tìm điểm dừng (bài toán đơn giản nhất). Thông thường điểm dừng là S(1), S(0),…

Bước 4. Viết code

+ Điểm dừng trước.

+ Gọi đệ quy sau.

LƯU Ý : trên đây chỉ là các bước luyện tập với những bài toán đệ quy đơn giản. Đối với những bài toán phức tạp hơn thì cần nhiều bước hơn, cần phải có sự tư duy cao.

Luyện tập thử ***8 bài toán ví dụ*** luôn nhé !!!

3b. Những bài toán đơn giản

Bài toán 1 : vẫn là bài toán quen thuộc .

Tính S(n) = 1 + 2 + 3 + .. + n.

Bước 1. Viết S(n) và S(n-1).

S(n) = 1 + 2 + 3 + … + (n-1) + n

S(n-1) = 1 + 2 + 3 + … + (n-1)

Bước 2. Biểu diễn S(n) theo S(n-1).

S(n) = S(n-1) + n

Bước 3. Tìm điểm dừng (bài toán đơn giản nhất).

S(1) = 1

Bước 4.

Dòng	Mã

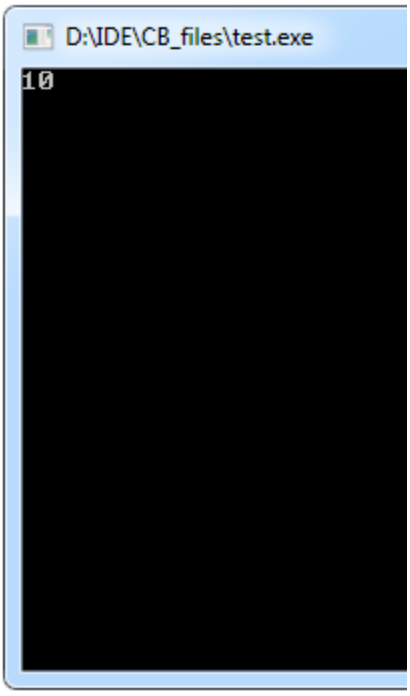
1
2
3
4
5
6
	`int S (int n)`
	`{`
	`  if (n == 1) return 1;`
	
	`  return S(n-1) + n;`
	`}`

Lưu ý quan trọng: đáng lẽ dòng code thứ 3 nên là `if (n <= 1)` cho an toàn (tránh trường hợp đứa nào ***ngu quá*** nhập số âm thì chết).

7 ví dụ sau bạn cũng lưu ý nhé.

Chạy thử nhé

```
1   #include <stdio.h>
2   #include <conio.h>
3
4   int S (int n)
5   {
6       if (n <= 1)
7           return 1;
8
9       return S(n-1) + n;
10  }
11
12  int main()
13  {
14      //S(n) = 1 + 2 + 3 + ... + n
15
16      printf("%d", S(4));
17
18      _getch();
19      return 0;
20  }
21
```



Kết quả S(4) = 1 + 2 + 3 + 4 = 10. Chính xác. 

Bài toán 2 : Tính $P(n) = 2^n$.

Bước 1.

$P(n) = \underbrace{2 * 2 * 2 * \dots * 2 * 2}_{\text{n số 2}}$

$P(n-1) = \underbrace{2 * 2 * 2 * \dots * 2}_{\text{n-1 số 2}}$

Bước 2.

$P(n) = P(n-1) * 2$

Bước 3.

$P(0) = 1$

```
1  #include <stdio.h>
2  #include <conio.h>
3
4  int P (int n)
5  {
6      if (n <= 0)
7          return 1;
8
9      return P(n-1) * 2;
10 }
11
12 int main()
13 {
14     //P(n) = 2^n = 2 mu~ n
15
16     printf("%d", P(4));
17
18     _getch();
19     return 0;
20 }
21
```

```
D:\IDE\CB_files\test.exe
16
```

Bước 4.

Dòng	Mã
1	`int P (int n)`
2	`{`
3	`    if (n <= 0)`
4	`        return 1;`
5	
6	`    return P(n-1) * 2;`
7	`}`

Bài toán 3 : Tính P(x,n) = x^n (x và n là số nguyên, n ≥ 0).

Đây là bài toán mở rộng của bài toán số 2 ở trên.

Hãy tự suy nghĩ 3 bước đầu nhé.

Bước 4.

Dòng	Mã
1	`int P (int x, int n)`
2	`{`
3	`    if (n <= 0)`
4	`        return 1;`
5	
6	`    return P(x, n-1) * x;`
7	`}`

Bài toán 4 :

Tính tổng $T(n) = \frac{1}{2} + \frac{2}{3} + \frac{3}{4} + \cdots + \frac{n}{n+1}$

Bài toán này cũng không quá khó, bạn xem thử code rồi suy nghĩ xem nhé.

Bước 4.

Dòng	Mã
1	`float T (int n)`
2	`{`
3	`    if (n <= 1)`
4	`        return 0.5;`
5	
6	`    return T(n-1) + (float)n/(n+1);`
7	`}`

Bài toán 5 :

Viết hàm kiểm tra xem mảng a có toàn số dương hay không.

a[0]	a[1]	a[2]	…	a[n-2]	a[n-1]

Bước 1 và 2.

Ta gộp chung cả 2 bước vì ta khó tưởng tượng được.

- Giả sử ta muốn mảng a có 4 phần tử đều là số dương.
  Tức là từ a[0] đến a[3] đều là số dương.
  Hay nói cách khác: từ a[0] đến a[2] là số dương, và a[3] > 0.

- Muốn a[0] đến a[2] là số dương,
  Thì a[0] đến a[1] là số dương, và a[2] > 0.
  …

⇨ ToànSốDương (a[0] đến a[n-1]) = ToànSốDương (a[0] đến a[n-2]) && a[n-1] > 0.
⇨ **ToànSốDương (n)** = **ToànSốDương (n-1)** && **a[n-1] > 0**.

Bước 3.
Khi mảng a chỉ có 1 phần tử a[0], ta dễ dàng kiểm tra được a[0] là số dương hay không.

Bước 4.

Dòng	Mã

```
1  int ToanSoDuong (int a[], int n)
2  {
3       if (n <= 1)
4            return (a[0] > 0);
5
6       return ToanSoDuong(a,n-1) && (a[n-1] > 0);
7  }
```

3c. Những bài toán phức tạp hơn

Bài toán 6 : tính R(n) = $\sqrt{n + \sqrt{(n-1) + \sqrt{(n-2) + \sqrt{\ldots + \sqrt{1}}}}}$ (n dấu căn)

Bước 1 và bước 2.

Rõ ràng ta thấy nếu viết R(n) và R(n-1) sẽ rất khó tìm thấy mối liên hệ giữa chúng. Đây là bài toán khó hơn :D Ta cứ thử viết đại đi.

R(2) = $\sqrt{2 + \sqrt{1}}$

R(1) = $\sqrt{1}$

⇨ R(n) = $\sqrt{n + R(n-1)}$

Đây chỉ là ta dự đoán, ta cần kiểm tra lại bằng máy tính bỏ túi. Kiểm tra lại thì ta thấy dự đoán đúng (khỏi cần chứng minh = Toán học :v).

Bước 3.

R(1) = $\sqrt{1}$ = 1

Bước 4. (nhớ khai báo thư viện math.h nhé)

Dòng	Mã
1	`float R (int n)`
2	`{`
3	`    if (n <= 1) return 1;`
4	
5	`    return sqrt(n + R(n-1));`
6	`}`

Bạn nên xem lại ví dụ số 3 trong mục "2c. Các ví dụ minh họa" để xem lược đồ gọi đệ quy nhé.

Bài toán 7 : Viết hàm kiểm tra xem N có phải là số có toàn chữ số lẻ hay không.

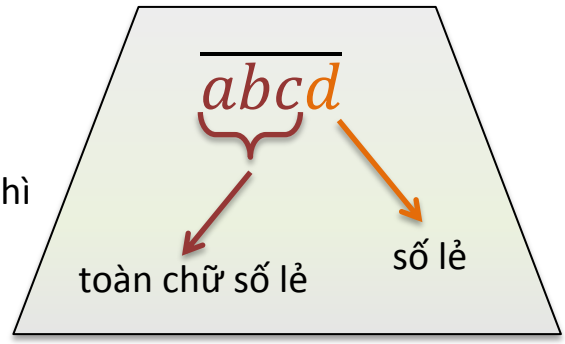
Đụng tới các bài toán về chữ số, ta liền nhớ về các chiêu thức div 10 và mod 10.

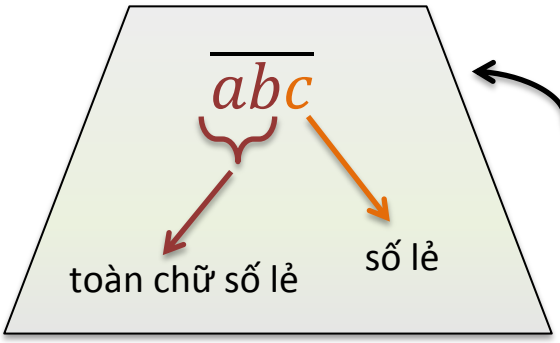
- Nếu N có dạng $\overline{abcd}$ thì
  + N / 10 = $\overline{abc}$
  + N % 10 = d

Bước 1 và 2.

Giả sử số N có dạng $\overline{abcd}$

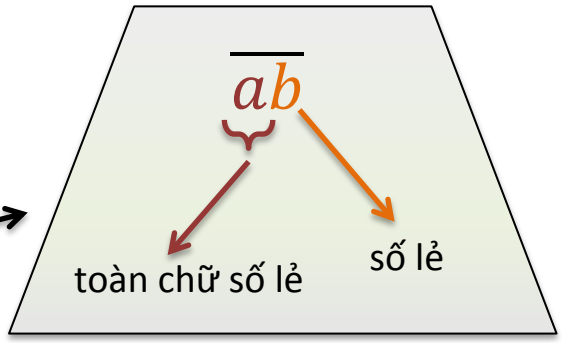
Muốn $\overline{abcd}$ có toàn chữ số lẻ thì





Muốn $\overline{abc}$ có toàn chữ số lẻ thì

Muốn $\overline{ab}$ có toàn chữ số lẻ thì



Và cuối cùng muốn $\bar{a}$ có toàn chữ số lẻ thì a là số lẻ, ta dễ dàng xét được.

Đến đây ta rút ra được : ***N toàn chữ số lẻ ⇔ (N%10 lẻ)*** *và* ***(N/10 toàn chữ số lẻ)***.

Bước 3.

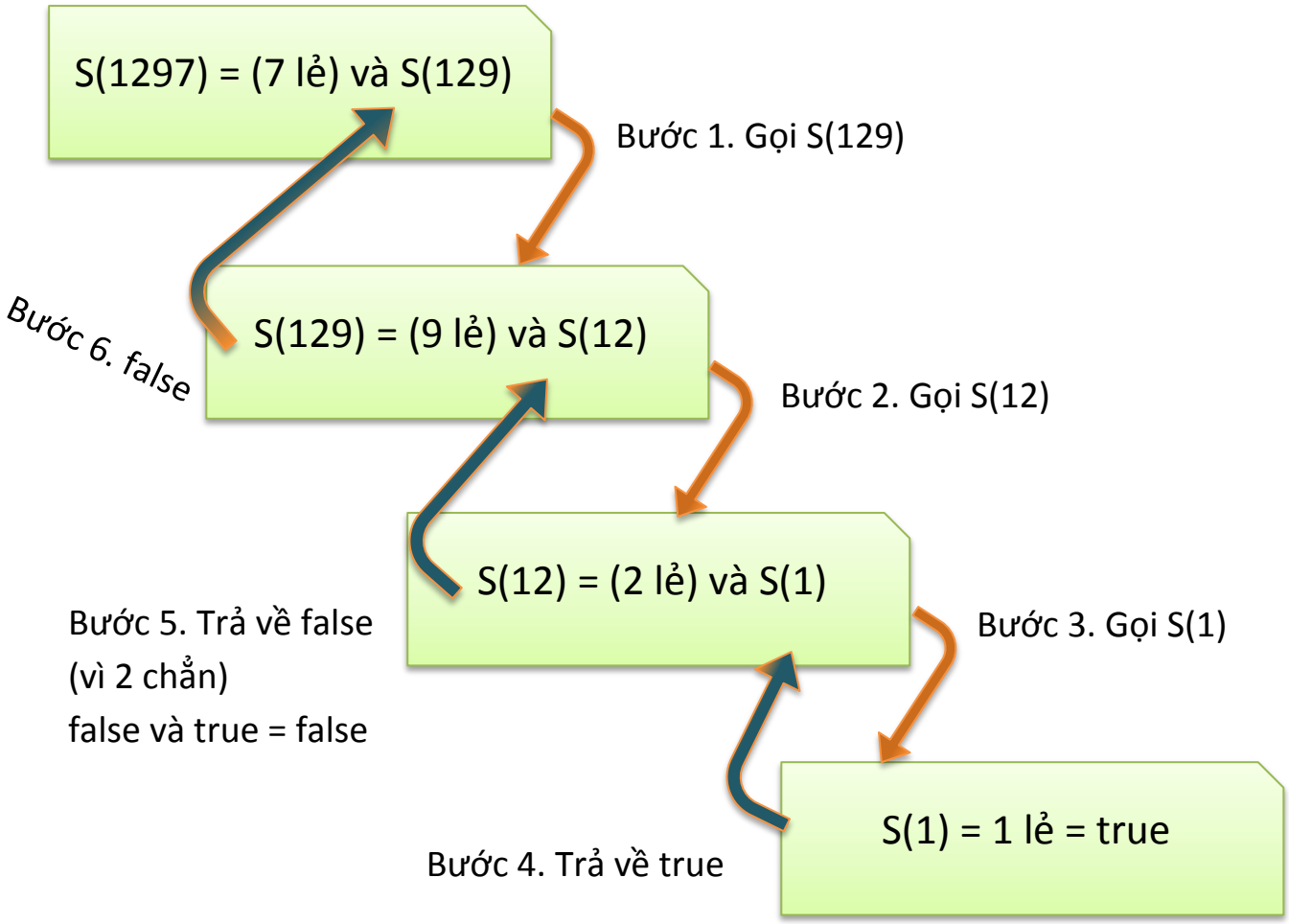
Bài toán đơn giản nhất là khi N có 1 chữ số, ta dễ dàng xét được N là số lẻ hay không.

Bước 4.

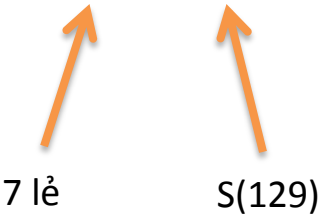
Dòng	Mã
1	`int ToanChuSoLe (int N)`
2	`{`
3	`     if (N < 10)`
4	`          return (N % 2 == 1);`
5	
6	`     int HangDonVi = N % 10;`
7	`     return (HangDonVi % 2 == 1) && ToanChuSoLe(N/10);`
8	`}`

Lược đồ phác họa các bước đi của bài toán 7 như sau :

Giả sử N = 1297. Để cho ngắn gọn, ta hiểu hàm ToanChuSoLe là hàm S.



Và cuối cùng là bước 7: S(1297) = true và false = false.

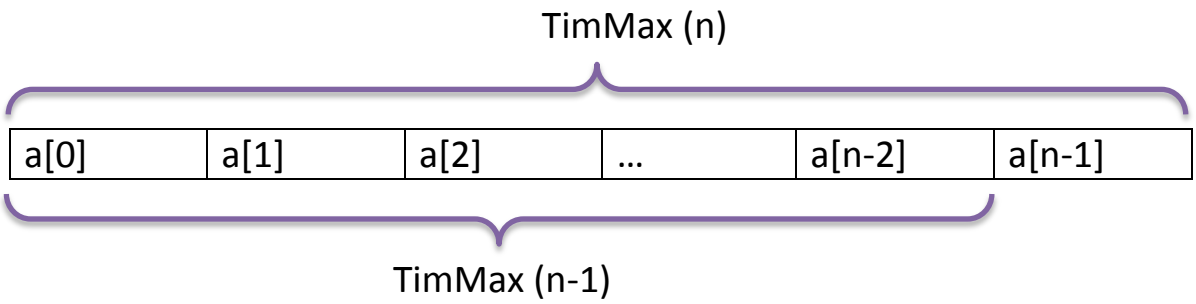


Kết luận : 1297 không phải là số có toàn chữ số lẻ.

Bài toán 8 : Tìm số lớn nhất mảng 1 chiều có n phần tử.

Ban đầu mình định để cho các bạn suy nghĩ bài này, nhưng mình thấy không ổn nên mình làm luôn. Bài này mình thấy cách trình bày hơi lan man, mong bạn chú ý kĩ.

Mảng a như sau



Bước 1.

TimMax (n) = số lớn nhất từ a[0] đến a[n-1]

TimMax (n-1) = số lớn nhất từ a[0] đến a[n-2] = Q

Bước 2.

Diễn dịch lại, ta có thể suy ra TimMax(n) = max (a[n-1], Q)

Công thức truy hồi sẽ là

TimMax(n) = số lớn nhất trong 2 số a[n-1] và TimMax(n-1)

Bước 3.

Khi n = 1, mảng a chỉ có 1 phần tử duy nhất là a[0], nên a[0] cũng là max.

TimMax(1) = a[0]

Bước 4.

Dòng	Mã
1	`int TimMax (int a[], int n)`
2	`{`
3	`    if (n <= 1)`
4	`        return a[0];`
5	
6	`    int max2 = TimMax (a, n-1);`
7	
8	`    if (a[n-1] > max2)`
9	`        return a[n-1];`
10	
11	`    return max2;`
12	`}`

Nói thêm : thật ra các bạn có thể code ngắn gọn hơn như sau

Dòng	Mã ngắn gọn hơn (không khuyến khích sử dụng)
1	`int TimMax (int a[], int n)`
2	`{`
3	`    if (n <= 1)`
4	`        return a[0];`
5	
6	`    if (a[n-1] > TimMax(a, n-1))`
7	`        return a[n-1];`
8	
9	`    return TimMax(a, n-1);`
10	`}`

Nhưng chúng ta phải nhớ 1 khuyết điểm của đệ quy là sự **bùng nổ tổ hợp**.

Tưởng tượng mảng có 50 phần tử = {49, 48, …, 1, 0}. Như vậy trường hợp xui xẻo nhất đã xuất hiện: hàm TimMax luôn được gọi 2 lần vì a[i] luôn > a[i+1]. Khi ấy tổng số lần hàm TimMax được gọi là **$2^{50} - 1$ lần**, một con số rất lớn…(treo máy).

Trong khi đó, nếu bạn sử dụng code mẫu của mình ở trên, thì chỉ mất **50 lần** gọi hàm max mà thôi.

“ **”**

Sự chênh lệch vô cùng lớn sẽ tạo ra đẳng cấp khác biệt.

4. Bài tập

Tất cả 8 ví dụ mình làm ở trên, đều thuộc dạng ***đệ quy tuyến tính***.

Nếu có thời gian, bạn nên luyện tập với bài toán sau :

a) Tính tổng $T(n) = 1^2 + 2^2 + 3^2 + \ldots + n^2$.

b) Tính tổng $Q(n) = \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \cdots + \frac{1}{2^n}$

c) Tìm số Fibonacci thứ n. Đây là loại đệ quy nhị phân.

d) Tìm ước chung lớn nhất của 2 số (2 cách : trừ liên tiếp và chia Euclide).

e) Đếm các chữ số lẻ của số N.

f) Ta đã quen thuộc với bài toán S(n) = 1 + 2 + 3 + … + n.

Dòng	Mã

1
2
3
4
5
6
	`int S (int n)`
	`{`
	`  if (n == 1) return 1;`
	
	`  return S(n-1) + n;`
	`}`

Có một cách giải khác bằng đệ quy. Để tính tổng 1+2+…+10 ta gọi hàm S(1,10).

Dòng	Mã

1
2
3
4
5
6
	`int S (int i, int n)`
	`{`
	`  if (i <= n) return i;`
	
	`  return i + S(i+1,n);`
	`}`

Cách giải này có “bình thường” như những gì ta đã làm ? Khi nào ta mới giải được bằng cách “quái lạ” này ?

g) (nâng cao) Mảng tăng dần.

Cho mảng a có 3 phần tử. Giá trị mỗi phần tử nằm từ 0 đến 9 ($0 \le a[i] \le 9$).

Mảng a được gọi là mảng tăng khi $a[i-1] \le a[i]$ với $1 \le i < n$.

Ví dụ a = {1,2,3}, a = {2,5,8},…

Yêu cầu : đếm số lượng mảng a tăng dần (có 3 phần tử).

Đây là bài giải bình thường. *Code chưa được tốt, chưa được tối ưu*.

u chính là a[0], v là a[1] và t là a[2].

Dòng	Mã
1	`int Dem_Mang_Tang()`
2	`{`
3	`    int dem = 0;`
4	
5	`    for (int u = 0; u <= 9; u++)`
6	`    for (int v = 0; v <= 9; v++)`
7	`    for (int t = 0; t <= 9; t++)`
8	`        if (u <= v && v <= t)`
9	`        {`
10	`            dem++;`
11	`        }`
12	
13	`    return dem;`
14	`}`

Giả sử mảng a không có 3 phần tử mà có ***n phần tử*** thì cách làm bình thường có giải được không ?

Viết hàm đệ quy giải bài toán trên.

5. Lời kết

Trên đây chỉ là những điều cơ bản nhất về đệ quy.

Mình cũng mất rất là nhiều thời gian để làm tài liệu này trong khi mình có thể vui vẻ đi ăn chơi Tết thoải mái. Chỉ có điều mình sợ rằng tài liệu chưa được tốt, có khi đệ quy quá khó hiểu sẽ làm cho các bạn dễ bị chết. Sau này các bạn sẽ còn tiếp cận với những bài toán đệ quy đẳng cấp cao hơn. Hi vọng nếu bạn vững được gần hết các ý trong tài liệu này thì bạn sẽ có một cái nền vững chắc.

Mình cũng mong rằng nếu nhiệt tình hơn, bạn có thể đóng góp ý kiến về cách trình bày của tài liệu, các ví dụ,…để cho tài liệu ngày một hoàn thiện hơn.

Cảm ơn các bạn đã sử dụng tài liệu này.

(Tài liệu được cập nhật ngày 27 tháng 2 năm 2015).

Nguyễn Trung Thành.

Đại học Khoa học Tự nhiên (HCMUS).

https://www.facebook.com/abcxyztcit

Email: abcxyz999.ntt@gmail.com