



TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG



Thuật toán ứng dụng

Nguyễn Khánh Phương

Computer Science department
School of Information and Communication technology
E-mail: phuongnk@soict.hust.edu.vn

Nội dung khóa học

Chương 1. Các cấu trúc dữ liệu và thư viện

Chương 2. Kỹ thuật đệ quy và nhánh cận

Chương 3. Chia nhỏ trị

Chương 4. Quy hoạch động

Chương 5. Các thuật toán trên đồ thị và ứng dụng

Chương 6. Các thuật toán xử lý xâu và ứng dụng

Chương 7. Lớp bài toán NP-dài đủ

NGUYỄN KHÁNH PHƯƠNG
KHMT – SOICT - DHBK HN

Nội dung chương 3

1. Sơ đồ chung của thuật toán
2. Các ví dụ minh họa
 - 2.1. Sắp xếp trộn và sắp xếp nhanh
 - 2.2. Tìm kiếm nhị phân
 - 2.3. Nhân số nguyên
 - 2.4. Nhân ma trận
 - 2.5. Lũy thừa nhị phân
 - 2.6. Từ Fibonacci

NGUYỄN KHÁNH PHƯƠNG
KHMT – SOICT - DHBK HN

Nội dung chương 3

1. Sơ đồ chung của thuật toán
2. Các ví dụ minh họa
 - 2.1. Sắp xếp trộn và sắp xếp nhanh
 - 2.2. Tìm kiếm nhị phân
 - 2.3. Nhân số nguyên
 - 2.4. Nhân ma trận
 - 2.5. Lũy thừa nhị phân
 - 2.6. Từ Fibonacci

NGUYỄN KHÁNH PHƯƠNG
KHMT – SOICT - DHBK HN

1. Sơ đồ chung của thuật toán

- Chia để trị (Divide and Conquer): thủ tục gồm 3 thao tác:
 - Divide: Phân rã bài toán đã cho thành **bài toán cùng dạng với kích thước nhỏ hơn** (gọi là **bài toán con**) S_1, S_2, \dots
 - Conquer: Giải các bài toán con một cách **dễ quy**
 - Combine: Tổng hợp lời giải của các bài toán con S_1, S_2, \dots để thu được lời giải của bài toán ban đầu S
- Nếu bài toán con là đủ nhỏ có thể dễ dàng giải được, thì ta tiến hành giải trực tiếp, nếu không: bài toán con lại được giải bằng cách áp dụng đệ quy thủ tục trên (tức là **lại tiếp tục chia nó thành các bài toán nhỏ hơn**). Do đó, các thuật toán chia để trị là các thuật toán đệ quy => để phân tích độ phức tạp có thể sử dụng công thức đệ quy

NGUYỄN KHÁNH PHƯƠNG 5
KHMT – SOICT - DHBK HN

1. Sơ đồ thuật toán chia để trị

Đề có được mô tả chi tiết của thuật toán chia để trị chúng ta cần phải xác định:

- Kích thước tối hạn n_0 (bài toán với kích thước nhỏ hơn n_0 sẽ không cần chia nhỏ)
- Kích thước của mỗi bài toán con trong cách chia
- Số lượng các bài toán con như vậy
- Thuật toán tổng hợp lời giải của các bài toán con.

Gọi $T(n)$ là thời gian tính của thuật toán

```
procedure D-and-C (n)
begin
  if n ≤ n0 then
    Giải bài toán một cách trực tiếp
  else
    begin
      Chia bài toán thành a bài toán con kích thước n/b
      for (mỗi bài toán trong a bài toán con) đo D-and-C(n/b);
      Tổng hợp lời giải của a bài toán con đê thu được lời giải của bài toán gốc;
    end;
end;
```

trong đó a là hằng số

$$T(n) = \begin{cases} c, & n \leq n_0, \\ aT(n/b) + D(n) + C(n), & n > n_0 \end{cases}$$

NGUYỄN KHÁNH PHƯƠNG 6
KHMT – SOICT - DHBK HN

Ví dụ.

```
procedure D-and-C(int n)
begin
  if (n == 0)  return;
  D-and-C(n/2);
  D-and-C(n/2);
  for (int i = 0 ; i < n; i++)
  begin
    //Thực hiện các thao tác thời gian cỡ hằng số
  end;
end;
```

Độ phức tạp tính toán của thuật toán chia để trị này là gì ?

Trả lời: $T(n) = 2T(n/2) + n$

Giải ta được $T(n) = O(n\log_2 n)$ (hoặc có thể sử dụng định lý Thợ)

7

Định lý Thợ (Master Theorem)

- Cung cấp công cụ để giải công thức đệ quy dạng

$$T(n) = a T(n/b) + f(n)$$

Các giả thiết:

$a \geq 1$ và $b \geq 2$ là các hằng số

$f(n)$ hàm đa thức

$T(n)$ được xác định với đối số nguyên không âm

Ta dùng n/b thay cho cả $\lfloor n/b \rfloor$ lẫn $\lceil n/b \rceil$

NGUYỄN KHÁNH PHƯƠNG
 KHMT – SOICT - DHBK HN

Định lý Thợ (Master Theorem)

- Xét công thức đệ quy $T(n) = a T(n/b) + f(n)$ thoả mãn các điều kiện
đã nêu, khi đó có thể đánh giá tiệm cận $T(n)$ như sau:
- Nếu $f(n) = O(n^{\log_b a - \varepsilon})$ đối với hằng số $\varepsilon > 0$ nào đó, thì $T(n) = \Theta(n^{\log_b a})$
 - Nếu $f(n) = \Theta(n^{\log_b a})$, thì $T(n) = \Theta(n^{\log_b a} \log n)$
 - Nếu $f(n) = \Omega(n^{\log_b a + \varepsilon})$ với hằng số $\varepsilon > 0$ nào đó, và nếu $a f(n/b) \leq c f(n)$
với hằng số $c < 1$ và với mọi n đủ lớn, thì $T(n) = \Theta(f(n))$.

Định lý Thợ rút gọn (Simplified Master Theorem):

Giả sử $a \geq 1, b \geq 2, c > 0$ là các hằng số. Xét $T(n)$ là công thức đệ qui $T(n) = a T(n/b) + f(n)$
xác định với $n \geq 0$; và $f(n) = c^k n^k$, hay nói cách khác $f(n) = \Theta(n^k)$ với $k \geq 0$

- Nếu $a > b^k$, thì $T(n) = \Theta(n^{\log_b a})$
- Nếu $a = b^k$, thì $T(n) = \Theta(n^k \log n)$
- Nếu $a < b^k$, thì $T(n) = \Theta(n^k)$

Master Theorem: Pitfalls

Định lý Thợ rút gọn (Simplified Master Theorem):

Giả sử $a \geq 1, b \geq 2, c > 0$ là các hằng số. Xét $T(n)$ là công thức đệ qui $T(n) = a T(n/b) + f(n)$
xác định với $n \geq 0$. và $f(n) = c^k n^k$ hay nói cách khác $f(n) = \Theta(n^k)$

- Nếu $a > b^k$, thì $T(n) = \Theta(n^{\log_b a})$.
- Nếu $a = b^k$, thì $T(n) = \Theta(n^k \log n)$.
- Nếu $a < b^k$, thì $T(n) = \Theta(n^k)$.

- Ta không thể sử dụng định lý thợ nếu:

- $T(n)$ không phải là hàm đơn điệu, ví dụ $T(n) = \sin(n)$
- $f(n)$ không phải là hàm đa thức, ví dụ $T(n) = 2T(n/2) + 2^n$
- b không được biểu diễn bởi hằng số, ví dụ $T(n) = T(\sqrt{n})$

NGUYỄN KHÁNH PHƯƠNG
KHMT – SOICT- DHBK HN

Master Theorem: Ví dụ 1

- Cho $T(n) = T(n/2) + \frac{1}{2} n^2 + n$. Xác định giá trị các tham số?

$$\begin{aligned} a &= 1 \\ b &= 2 \\ k &= 2 \end{aligned}$$

Do đó, điều kiện nào thỏa mãn?

$$1 < 2^2 \rightarrow \text{điều kiện 3}$$

- Vậy $T(n) \in \Theta(n^2) = \Theta(n^2)$

Định lý Thợ rút gọn (Simplified Master Theorem):

Giả sử $a \geq 1, b \geq 2, c > 0$ là các hằng số. Xét $T(n)$ là công thức đệ qui $T(n) = a T(n/b) + f(n)$
xác định với $n \geq 0$; và $f(n) = c^k n^k$, hay nói cách khác $f(n) = \Theta(n^k)$ với $k \geq 0$

- Nếu $a > b^k$, thì $T(n) = \Theta(n^{\log_b a})$
- Nếu $a = b^k$, thì $T(n) = \Theta(n^k \log n)$
- Nếu $a < b^k$, thì $T(n) = \Theta(n^k)$

NGUYỄN KHÁNH PHƯƠNG
KHMT – SOICT- DHBK HN

Master Theorem: Ví dụ 2

- Cho $T(n) = 2 T(n/4) + \sqrt{n} + 42$. Xác định giá trị các tham số?

$$\begin{aligned} a &= 2 \\ b &= 4 \\ k &= 1/2 \end{aligned}$$

Do đó, điều kiện nào thỏa mãn?

$$2 = 4^{1/2} \rightarrow \text{điều kiện 2 thỏa mãn}$$

- Vậy $T(n) \in \Theta(n^d \log n) = \Theta(\log n \sqrt{n})$

Định lý Thợ rút gọn (Simplified Master Theorem):

Giả sử $a \geq 1, b \geq 2, c > 0$ là các hằng số. Xét $T(n)$ là công thức đệ qui $T(n) = a T(n/b) + f(n)$
xác định với $n \geq 0$; và $f(n) = c^k n^k$, hay nói cách khác $f(n) = \Theta(n^k)$ với $k \geq 0$

- Nếu $a > b^k$, thì $T(n) = \Theta(n^{\log_b a})$
- Nếu $a = b^k$, thì $T(n) = \Theta(n^k \log n)$
- Nếu $a < b^k$, thì $T(n) = \Theta(n^k)$

Master Theorem: Ví dụ 3

- Cho $T(n) = 3T(n/2) + (3/4)n + 1$. Xác định giá trị các tham số?

$$\begin{aligned} a &= 3 \\ b &= 2 \\ k &= 1 \end{aligned}$$

Do đó, điều kiện áp dụng là?

$3 > 2^1$, \rightarrow điều kiện 2 thỏa mãn

- Vậy $T(n) \in \Theta(n^{\log_b a}) = \Theta(n^{\log_2 3})$

- Chú ý $\log_2 3 \approx 1.584\dots$, liệu ta có thể nói $T(n) \in \Theta(n^{1.584}) \dots$????

NO, vì $\log_2 3 \approx 1.584\dots$ và $n^{1.584} \notin \Theta(n^{1.5849})$

Dịnh lý Thay thế rút gọn (Simplified Master Theorem):

Giả sử $a \geq 1$, $b \geq 2$, $c > 0$ là các hằng số. Xét $T(n)$ là công thức đệ quy
 $T(n) = aT(n/b) + f(n)$
xác định với $n \geq 0$; và $f(n) = c^n n^k$, hay nói cách khác $f(n) = \Theta(n^k)$ với $k \geq 0$
1. Nếu $a > b^k$, thì $T(n) = \Theta(n^{\log_b a})$
2. Nếu $a = b^k$, thì $T(n) = \Theta(n^k \log n)$
3. Nếu $a < b^k$, thì $T(n) = \Theta(n^k)$

Nội dung

1. Sơ đồ chung của thuật toán

2. Các ví dụ minh họa

2.1. Sắp xếp trộn (Merge sort) và sắp xếp nhanh (Quick sort)

2.2. Tìm kiếm nhị phân

2.3. Nhân số nguyên

2.4. Nhân ma trận

2.5. Lũy thừa nhị phân

2.6. Tứ Fibonacci

NGUYỄN KHÁNH PHƯƠNG 14
KHMT – SOICT - DHBK HN

Chia để trị (Divide and Conquer)

Idea Merge sort:

- Divide:** Chia dãy n phần tử thành hai bài toán con với $n/2$ phần tử
- Conquer:** Sắp xếp nửa trái và nửa phải một cách đệ quy sử dụng sắp xếp trộn. Nếu dãy chỉ có 1 phần tử, thì không làm gì cả vì nó đã được sắp xếp rồi.
- Combine:** trộn (merge) hai nửa đã được sắp xếp lại để thu được dãy n phần tử đã sắp xếp

Idea Quick sort:

- Divide:** chia dãy n phần tử thành hai tập: tập chứa các phần tử “nhỏ” sẽ nằm ở nửa đầu dãy và tập chứa các phần tử “lớn” sẽ nằm ở nửa cuối dãy
- Conquer:** Sắp xếp hai tập này một cách đệ quy sử dụng Quicksort. Nếu dãy chỉ có 1 phần tử, thì không làm gì cả vì nó đã được sắp xếp rồi.
- Combine:** dãy thu được đã sắp xếp mà không cần phải làm thêm thao tác gì

NGUYỄN KHÁNH PHƯƠNG 15
KHMT – SOICT - DHBK HN

Sắp xếp trộn (Merge sort)

Idea Merge sort:

- Divide: Chia dãy n phần tử thành hai bài toán con với $n/2$ phần tử
- Conquer: Sắp xếp nửa trái và nửa phải một cách đệ quy sử dụng sắp xếp trộn. Nếu dãy chỉ có 1 phần tử, thì không làm gì cả vì nó đã được sắp xếp rồi.
- Combine: trộn (merge) hai nửa đã được sắp xếp lại để thu được dãy n phần tử đã sắp xếp

MergeSort(A , l , r)

```
if  $l < r$  then
     $m \leftarrow \lfloor (l + r)/2 \rfloor$ 
    MergeSort( $A$ ,  $l$ ,  $m$ )
    MergeSort( $A$ ,  $m + 1$ ,  $r$ )
    MERGE( $A$ ,  $l$ ,  $m$ ,  $r$ )
endif
```

Lệnh gọi thực hiện thuật toán cho mảng A gồm n phần tử: MergeSort(A , 0, $n-1$);



Thời gian tính của sắp xếp trộn Merge Sort:

- Chắc chắn m như là giá trị trung bình của l và r : $O(1)$
- Tổng giải đệ quy 2 bài toán con, mỗi bài toán kích thước $n/2 \Rightarrow 2T(n/2)$
- Tổ hợp: TRÔN (MERGE) trên các mảng con có n phần tử đòi hỏi thời gian $O(n)$:

$$T(n) = \begin{cases} O(1) & \text{nếu } n = 1 \\ 2T(n/2) + O(n) & \text{nếu } n > 1 \end{cases}$$

Suy ra: $T(n) = O(n \log n)$ (CM bằng qui nạp hoặc cây đệ quy)

Dịnh lý Thay thế rút gọn (Simplified Master Theorem):

Giả sử $a \geq 1$ và $b > 1$ là các hằng số. Xét $T(n)$ là công thức đệ quy

$$T(n) = aT(n/b) + c n^k$$

xác định với $n \geq 1$.

$$1. \text{ Nếu } a > b^k, \text{ thì } T(n) = \Theta(n^{\log_b a}).$$

$$2. \text{ Nếu } a = b^k, \text{ thì } T(n) = \Theta(n^k \log n).$$

$$3. \text{ Nếu } a < b^k, \text{ thì } T(n) = \Theta(n^k).$$

Merge sort: $a = 2$, $b = 2$
CASE 2 ($k = 1$) $\rightarrow T(n) = \Theta(n \log n)$

16

Sắp xếp nhanh (Quick sort)

1. **Neo đê qui** (Base case). Nếu dãy chỉ còn không quá một phần tử thì nó là dãy được sắp và trả lại ngay dãy này mà không phải làm gì cả.

2. **Chia** (Divide):

- Chọn một phần tử trong dãy và gọi nó là phần tử chốt p (pivot).
- Chia dãy đã cho ra thành hai dãy con: Dãy con trái (L) gồm những phần tử \leq phần tử chốt, còn dãy con phải (R) gồm các phần tử \geq phần tử chốt. Thao tác này được gọi là "Phân đoạn" (Partition).



3. **Trị** (Conquer): Lặp lại một cách đê qui thuật toán QuickSort đối với hai dãy con $L = A[\text{left} \dots k-1]$ và $R = A[k+1 \dots \text{right}]$.

4. **Tổng hợp** (Combine): Dãy được sắp xếp là $L \text{ } p \text{ } R$.

Ngược lại với Merge Sort, trong Quick Sort thao tác chia là phức tạp, nhưng thao tác tổng hợp lại đơn giản.

Điểm mấu chốt đê thực hiện Quick Sort chính là thao tác chia. Phụ thuộc vào thuật toán thực hiện thao tác này mà ta có các dạng Quick Sort cụ thể.

Sơ đồ tổng quát sắp xếp nhanh (Quick sort)

- Sơ đồ tổng quát của QS có thể mô tả như sau:

Quick-Sort($A, Left, Right$)

```
1. if ( $Left < Right$ ) {
2.     Pivot = Partition( $A, Left, Right$ );
3.     Quick-Sort( $A, Left, Pivot - 1$ );
4.     Quick-Sort( $A, Pivot + 1, Right$ );
5. }
```

Hàm **Partition($A, Left, Right$)** thực hiện chia $A[Left..Right]$ thành hai đoạn $A[Left..Pivot - 1]$ và $A[Pivot+1..Right]$ sao cho:

- Các phần tử trong $A[Left..Pivot - 1]$ là nhỏ hơn hoặc bằng $A[Pivot]$
- Các phần tử trong $A[Pivot+1..Right]$ là lớn hơn hoặc bằng $A[Pivot]$.

Lệnh gọi thực hiện thuật toán **Quick-Sort($A, 0, n-1$)**

18

Thao tác chia trong sắp xếp nhanh (Quick sort)

Chia (Divide):

- Chọn một phần tử trong dãy và gọi nó là phần tử chốt p (pivot).
- Chia dãy đã cho ra thành hai dãy con: Dãy con trái (L) gồm những phần tử \leq phần tử chốt, còn dãy con phải (R) gồm các phần tử \geq phần tử chốt. Thao tác này được gọi là "Phân đoạn" (Partition).



- Thao tác phân đoạn có thể cài đặt (tại chỗ) với thời gian $O(n)$.
- Việc chọn phần tử chốt có vai trò quyết định đối với hiệu quả của thuật toán. Tốt nhất nếu chọn được phần tử chốt là phần tử có giá trị trung bình trong danh sách (ta gọi phần tử như vậy là **trung vị/median**). Khi đó, sau $\log_2 n$ lần phân đoạn ta sẽ đạt tới danh sách với kích thước bằng 1. Tuy nhiên, điều đó rất khó thực hiện. Người ta thường sử dụng các cách chọn phần tử chốt sau đây:
 - Chọn phần tử trái nhất (đứng đầu) làm phần tử chốt.
 - Chọn phần tử phải nhất (đứng cuối) làm phần tử chốt.
 - Chọn phần tử đứng giữa danh sách làm phần tử chốt.
 - Chọn phần tử trung vị trong 3 phần tử đứng đầu, đứng giữa và đứng cuối làm phần tử chốt (Knuth).
 - Chọn ngẫu nhiên một phần tử làm phần tử chốt.

19

Sắp xếp nhanh (Quick sort)

Quick Sort có thời gian tính trung bình là $O(n \log n)$, tuy nhiên thời gian tính tồi nhất của nó lại là $O(n^2)$:

- **Worst case:**
 - Số phép so sánh cần thực hiện $\sim n^2/2$
- **Average Case:** số phép so sánh cần thực hiện $\sim 1.39n \log n$
 - Số phép so sánh cần thực hiện nhiều hơn ~39% so với sắp xếp trộn trong trường hợp tồi nhất
 - Nhưng nhanh hơn sắp xếp trộn vì ít phải di chuyển các phần tử

NGUYỄN KHÁNH PHƯƠNG
KHMT – SOICT - ĐHBK HN

Nội dung

1. Sơ đồ chung của thuật toán

2. Các ví dụ minh họa

2.1. Sắp xếp trộn (Merge sort) và sắp xếp nhanh (Quick sort)

2.2. Tìm kiếm nhị phân

2.3. Nhân số nguyên

2.4. Nhân ma trận

2.5. Lũy thừa nhị phân

2.6. Tứ Fibonacci

NGUYỄN KHÁNH PHƯƠNG 21
KHMT – SOICT - DHBK HN

2.2. Tìm kiếm nhị phân (Binary search)

Bài toán: Cho mảng số $A[0..n-1]$ được sắp xếp theo thứ tự tăng dần và số target. Cần tìm chỉ số i ($0 \leq i \leq n-1$) sao cho $A[i] = \text{target}$.

• Thuật toán chia để trị để giải bài toán được xây dựng dựa trên lập luận sau: Số target cho trước

- hoặc là bằng phần tử nằm ở vị trí ở giữa mảng A
- hoặc là nằm ở nửa bên trái (L) của mảng A
- hoặc là nằm ở nửa bên phải (R) của mảng A.
- hoặc là không có trong mảng A

```
int binary_search(const vector<int> &A , int low, int high , int target)
{
    if (low > high)  return -1;
    int middle = (low + high) / 2;
    if ( A [middle] == target)
        return middle;
    else if (target < A [middle])
        return binary_search (A, low , middle - 1, target);
    else if (target > A[middle])
        return binary_search (A, middle + 1, high , target);
}
```

Lệnh gọi hàm: `binary_search (A, 0, A.size()-1, target);`

22

2.2. Tìm kiếm nhị phân (Binary search)

```
int binary_search(const vector<int> &A , int low, int high , int target)
{
    if (low > high)  return -1;
    int middle = (low + high) / 2;
    if ( A [middle] == target)
        return middle;
    else if (target < A [middle])
        return binary_search (A, low , middle - 1, target);
    else if (target > A[middle])
        return binary_search (A, middle + 1, high , target);
}
```

Phân tích hiệu quả của thuật toán:

Gọi $T(n)$ là thời gian tính của thuật toán, ta có

$$T(1) = c$$

$$T(n) = T(n/2) + d$$

trong đó c, d là hằng số. Theo định lý lóp,

$$T(n) = \Theta(\log n)$$

NGUYỄN KHÁNH PHƯƠNG 23
KHMT – SOICT - DHBK HN

Giảm để trị (Decrease and Conquer)

• Trong nhiều ứng dụng, không nhất thiết phải chia bài toán cần giải thành nhiều bài toán con, mà chỉ giảm về một bài toán con kích thước nhỏ hơn.

• Kĩ thuật như vậy gọi là Giảm để trị

Ví dụ thông dụng nhất là tìm kiếm nhị phân trong mục 2.2

NGUYỄN KHÁNH PHƯƠNG 24
KHMT – SOICT - DHBK HN

Ví dụ 1: <https://codeforces.com/problemset/problem/279/B>

B. Books

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

When Valera has got some free time, he goes to the library to read some books. Today he's got t free minutes to read. That's why Valera took n books in the library and for each book he estimated the time he is going to need to read it. Let's number the books by integers from 1 to n . Valera needs a_i minutes to read the i -th book.

Valera decided to choose an arbitrary book with number i and read the books one by one, starting from this book. In other words, he will first read book number i , then book number $i+1$, then book number $i+2$ and so on. He continues the process until he either runs out of the free time or finishes reading the n -th book. Valera reads each book up to the end, that is, he doesn't start reading the book if he doesn't have enough free time to finish reading it.

Print the maximum number of books Valera can read.

Input
The first line contains two integers n and t ($1 \leq n \leq 10^5$; $1 \leq t \leq 10^9$) — the number of books and the number of free minutes Valera's got. The second line contains a sequence of n integers a_1, a_2, \dots, a_n ($1 \leq a_i \leq 10^7$), where number a_i shows the number of minutes that the boy needs to read the i -th book.

Output
Print a single integer — the maximum number of books Valera can read.

Examples

input	input
4 5	3 3
3 1 2 1	2 2 3
output	output
3	1

25

Ví dụ 1: <https://codeforces.com/problemset/problem/279/B>

- Duyệt toàn bộ: $O(n^2)$

– Chú ý điều kiện của đề bài: nếu Valera đọc quyền đầu tiên là quyền thứ i , thì bắt buộc quyền tiếp theo sẽ là $i+1$, rồi đến $i+2, \dots$ và kết thúc hoặc hết quyền thứ n hoặc nếu đọc quyền tiếp thì tổng thời gian đọc sẽ $> t$. Như vậy Valera có thể có tổng cộng n cách đọc sách sau:

- Cách 1: Bắt đầu đọc quyền số 1, và tiếp tục lần lượt các quyền khác cho đến khi nếu đọc quyền tiếp thì tổng thời gian đọc sẽ $> t$
- Cách 2: Bắt đầu đọc quyền số 2, và tiếp tục lần lượt các quyền khác cho đến khi nếu đọc quyền tiếp thì tổng thời gian đọc sẽ $> t$
-

– Duyệt cả n cách này, mỗi cách mất thời gian $O(n)$ để kiểm tra điều kiện thời gian. Lời giải của bài toán là số quyền lớn nhất mà Valera có thể đọc được sao cho tổng thời gian đọc $\leq t$.

Tăng tốc ???

NGUYỄN KHÁNH PHƯƠNG 26
KHMT - SOICT - DHBK HN

Ví dụ 1: <https://codeforces.com/problemset/problem/279/B>

- Tìm kiếm nhị phân: $O(n\log n)$

– Nếu dùng phương pháp duyệt toàn bộ, với mỗi giá trị i (tương ứng cách đọc bắt đầu từ quyền i), ta phải tính tổng thời gian Valera đọc các sách và kiểm tra xem tổng thời gian đó có vượt quá thời gian đã cho t hay không. Để tránh việc phải tính lại với mỗi giá trị i , ta sẽ dùng mảng sum :

- $sum[k] = sum[k-1] + time[k]$ với $k \geq 1$
- $sum[0] = time[0]$ với $k = 1$

Thời gian để tính mảng sum là $O(n)$.

Khi đó thời gian đọc từ quyền i đến hết quyền j là

$$sum[j] - sum[i] + time[i], \text{ với } j > i$$

for $i = 1$ to n

Với mỗi giá trị i ta cần tìm giá trị j đạt max sao cho:

$$i \leq j \leq n \quad \& \quad sum[j] - sum[i] + time[i] \leq t \rightarrow \text{tồn } O(n)$$

$O(n^2)$
 $O(n\log n)$

Tăng tốc???

↓

Tim kiém nhì phan: $O(\log n)$

27

Ví dụ 1: <https://codeforces.com/problemset/problem/279/B>

- Tìm kiếm nhị phân: $O(n\log n)$

for $i = 1$ to n

Với mỗi giá trị i ta cần tìm giá trị j đạt max sao cho:
 $i \leq j \leq n$ và $sum[j] - sum[i] + time[i] \leq t$

Tim kiém nhì phan:

Tìm trong mảng sum từ phần tử thứ i đến n , phần tử j thỏa mãn **điều kiện trên**

```
int result = INT_MIN;
for (int i = 1; i <= n; i++) //bat dau doc tu quyen thu i
{
    int j = binary_search(sum, i, n, t + sum[i] - time[i]);
    //j-i+1 la so sach doc duoc neu bat dau doc tu quyen thu i
    result = max(result, j - i + 1);
}
cout << result << endl;
```

BTVN: Viết hàm `int binary_search(int *sum, int low, int high, int target)` trả về chỉ số j thỏa mãn điều kiện đã nêu

28

Tìm kiếm nhị phân trên các số nguyên

- Xét hàm int $f(\text{unsigned int } x)$ nhận đầu vào là biến x nguyên không âm và trả về một số nguyên là hàm **tăng đơn điệu** (**monotically increasing function**), tức là với mọi giá trị $\text{unsigned int } x$ đầu vào ta luôn có: $f(x+1) > f(x)$.

Yêu cầu: Tim giá trị n sao cho hàm f lần đầu tiên nhận giá trị lớn hơn 0: tức là $f(n-1), f(n-2), \dots, f(0) < 0$. Vì f là hàm đơn điệu tăng, nên $f(n) > 0$ thì $f(n+1), f(n+2), \dots > 0$:

1	0	1	...	n-1	n	n+1	n+2	...
$f(i)$	< 0	< 0		< 0	> 0	> 0	> 0	

Ví dụ: $f(x) = x^2 - 4x - 8$. Khi đó giá trị cần tìm là $n = x = 6$

- Cách giải đơn giản: $O(K*n)$
Duyệt toàn bộ lần lượt $n = 0, 1, 2, \dots$ với mỗi n , ta tính giá trị $f(n)$, dừng khi thu được $f(n) > 0$
- Tìm kiếm nhị phân: $O(K*\log n)$
với K là giá của việc tính giá trị hàm f

29

Tìm kiếm nhị phân trên các số nguyên

- Tìm kiếm nhị phân: $O(\log n)$:

n	0	1	...	n-1	n	n+1	n+2	...
$f(n)$	< 0	< 0		< 0	> 0	> 0	> 0	

Để áp dụng tìm kiếm nhị phân trên mảng, ta cần có chỉ số low và high :

`int binary_search(int *Arr, int low, int high, int target)`

Để tìm được giá trị cho low và high , ta sẽ tính lần lượt giá trị:

$f(i=0),$
 $f(i=1),$
 $f(i=2),$
 $f(i=4),$
 $f(i=8),$
 $f(i=16),$
 $\dots,$
 $f(i=\text{high})$

i *= 2: giá trị i nhân đôi sau mỗi vòng lặp

với $f(\text{high})$ là giá trị đầu tiên mà hàm $f > 0$.

Ta sẽ áp dụng tìm kiếm nhị phân với biến high là giá trị vừa tìm được, và $\text{low} = \text{high}/2$

30

Tìm kiếm nhị phân trên các số nguyên

Ví dụ: Cho hàm $f: \{0,..,n-1\} \rightarrow \{\text{T, F}\}$ thỏa mãn điều kiện nếu $f(i) = \text{T}$ thì $f(j) = \text{T}$ với mọi $j > i$

i	0	1	...	j-1	j	j+1	j+2	...	n-1
$f(i)$	F	F		F	T	T	T		T

Trong khoảng 0 đến $n-1$, tìm giá trị j sao cho hàm f lần đầu tiên nhận giá trị T : tức là $f(0), f(1), \dots, f(j-1)$ đều = T.

Giải: Áp dụng tìm kiếm nhị phân: $O(K*\log n)$

với K là giá của việc tính giá trị hàm f

```
int low = 0;
int high = n-1;

while (low < high)
{
    int mid = (low + high)/2;
    if (f(mid)==T) high = mid;
    else low = mid+1;
}

if (low == high && f(low) == T)
    cout<<"Chỉ số nhỏ nhất tìm được là "<<low;
else
    cout<<"Không tồn tại chỉ số thỏa mãn điều kiện";
```

31

Tìm kiếm nhị phân trên các số nguyên

Cho hàm $f: \{0,..,n-1\} \rightarrow \{\text{T, F}\}$ thỏa mãn điều kiện nếu $f(i) = \text{T}$ thì $f(j) = \text{T}$ với mọi $j > i$

i	0	1	...	j-1	j	j+1	j+2	...	n-1
$f(i)$	F	F		F	T	T	T		T

Trong khoảng 0 đến $n-1$, tìm giá trị j sao cho hàm f lần đầu tiên nhận giá trị T : tức là $f(0), f(1), \dots, f(j-1)$ đều = T.

Giải: Áp dụng tìm kiếm nhị phân: $O(K*\log n)$ với K là giá của việc tính giá trị hàm f

Ví dụ ứng dụng: tìm vị trí của x trong mảng A đã sắp xếp theo thứ tự tăng dần

Trả lời: Dùng tìm kiếm nhị phân như ở trên, với hàm f là:

```
bool f(int i)
{
    return (A[i] >= x);
}

int low = 0;
int high = n-1;

while (low < high)
{
    int mid = (low + high)/2;
    if (f(mid)==T) high = mid;
    else low = mid+1;
}

if (low == high && f(low) == T)
    cout<<"Chỉ số nhỏ nhất tìm được là "<<low;
else
    cout<<"Không tồn tại chỉ số thỏa mãn điều kiện";
```

32

Tìm kiếm nhị phân trên các số thực

Đây là phiên bản tổng quát hơn của tìm kiếm nhị phân.

- Cho hàm $f: [low, high] \rightarrow \{T, F\}$ thỏa mãn nếu $f(i) = T$ thì $f(j) = T$ với mọi $j > i$.
Yêu cầu: Trong khoảng low đến $high$, tìm số thực j nhỏ nhất sao cho hàm f lần đầu tiên nhận giá trị T (tức là: $f(j) = T$ càng sớm càng tốt).
- Nhận thấy, khi làm việc với số thực, khoảng giá trị $[low, high]$ có thể bị chia vỡ hạn lẩn. Vì vậy, thay vì tìm chính xác giá trị j , ta có thể tìm giá trị thực j' rất sát với lời giải đúng j , với sai số trong khoảng chấp nhận được, ví dụ sai số $\text{eps} = 2^{-30} (\approx 10^{-10})$.
 - Ta có thể làm được điều này trong thời gian $O(\log \frac{high-low}{\text{eps}})$ tương tự cách làm tìm kiếm nhị phân trên mảng:

```
int low = -1000.0;
int high = 1000.0;
double eps = 1e-10;

while (high - low > eps)
{
    double mid = (low + high)/2;
    if (f(mid)==T) high = mid;
    else low = mid;
}
cout<<low<<endl;
```

33

Tìm kiếm nhị phân trên các số thực

Đây là phiên bản tổng quát hơn của tìm kiếm nhị phân.

- Cho hàm $f: [low, high] \rightarrow \{T, F\}$ thỏa mãn nếu $f(i) = T$ thì $f(j) = T$ với mọi $j > i$.
Yêu cầu: Trong khoảng low đến $high$, tìm số thực j nhỏ nhất sao cho hàm f lần đầu tiên nhận giá trị T (tức là: $f(j) = T$ càng sớm càng tốt).

Ví dụ ứng dụng 1: Tìm căn bậc hai của x

Khi đó, hàm f sẽ là

```
bool f(double j)
{
    return j*j >= x;
}
```

Ví dụ ứng dụng 2: Tìm nghiệm của hàm $g(x) = 0$

```
bool f(double x)
{
    return g(x) >= 0.0;
}
```

34

Ví dụ 2: Problem C from NWERC 2006

- NWERC (ACM Northwestern European Programming Contest)

Problem C: Pie

My birthday is coming up and traditionally I'm serving pie. Not just one pie, no, I have a number N of them, of various tastes and of various sizes. F of my friends are coming to my party and each of them gets a piece of pie. This should be one piece of one pie, not several small pieces from many pies—this would be really messy. This piece can be one whole pie though.

My friends are very annoying and if one of them gets a bigger piece than the others, they start complaining. Therefore all of them should get equally sized (but not necessarily equally shaped) pieces, even if this leads to some pie going to waste (which is better than annoying friends). Of course, I want a piece of pie myself and it would also be nice if this piece also has the same size.

What is the largest possible piece size all of us can get? All the pies are cylindrical in shape and they all have the same height 1, but the radii of the pies can be different.

Input

One line with a positive integer: the number of test cases. Then for each test case:

- One line with two integers N and F with $1 \leq N, F \leq 10000$: the number of pies and the number of friends.
- One line with N integers r_i with $1 \leq r_i \leq 10000$: the radii of the pies.

Sample input	Sample output
3	25.1337
3 3	3.1416
4 3 3	50.2655
1 24	
5	
10 5	
1 4 2 3 4 5 6 5 4 2	

Nội dung

1. Sơ đồ chung của thuật toán

2. Các ví dụ minh họa

2.1. Sắp xếp trộn (Merge sort) và sắp xếp nhanh (Quick sort)

2.2. Tìm kiếm nhị phân

2.3. Nhân số nguyên

2.4. Nhân ma trận

2.5. Lũy thừa nhị phân

2.6. Từ Fibonacci

For each test case, output one line with the largest possible volume V such that me and my friends can all get a pie piece of size V . The answer should be given as a floating point number with an absolute error of at most 10^{-3} .

NGUYỄN KHÁNH PHƯƠNG 35
KHMT – SOICT- DHBK HN

NGUYỄN KHÁNH PHƯƠNG 36
KHMT – SOICT- DHBK HN

2.3. Nhân số nguyên

$$\begin{array}{r}
 & 9 & 8 & 1 \\
 & 1 & 2 & 3 & 4 \\
 \hline
 & 3 & 9 & 2 & 4 \\
 & 2 & 9 & 4 & 3 \\
 1 & 9 & 6 & 2 \\
 9 & 8 & 1 \\
 \hline
 1 & 2 & 1 & 0 & 5 & 5 & 4
 \end{array}$$

- Nếu các nhân tử có n chữ số, thì thời gian cần thiết là $\Theta(n^2)$
- Có cách làm nào tốt hơn hay không?

NGUYỄN KHÁNH PHƯƠNG 37
KHMT - SOICT - DHBK HN

2.3. Nhân số nguyên

- Bài toán: Cho

$$x = x_{n-1} x_{n-2} \dots x_1 x_0 \text{ và}$$

$$y = y_{n-1} y_{n-2} \dots y_1 y_0$$

là 2 số nguyên không âm có n chữ số thập phân. Cần tính

$$z = z_{2n-1} z_{2n-2} \dots z_1 z_0$$

là biểu diễn với $2n$ chữ số thập phân của tích xy .

38

2.3. Nhân số nguyên: Thuật toán Karatsuba (1962)

- Ta có: $x = x_{n-1} x_{n-2} \dots x_1 x_0$ và $y = y_{n-1} y_{n-2} \dots y_1 y_0$

$$\Rightarrow x = x_{n-1} \times 10^{n-1} + x_{n-2} \times 10^{n-2} + \dots + x_1 \times 10^1 + x_0 \times 10^0$$

$$y = y_{n-1} \times 10^{n-1} + y_{n-2} \times 10^{n-2} + \dots + y_1 \times 10^1 + y_0 \times 10^0$$

- Vì thế: $z = z_{2n-1} z_{2n-2} \dots z_1 z_0 = x * y$

$$z = z_{2n-1} \times 10^{2n-1} + z_{2n-2} \times 10^{2n-2} + \dots + z_1 \times 10^1 + z_0 \times 10^0$$

$$= (x_{n-1} \times 10^{n-1} + x_{n-2} \times 10^{n-2} + \dots + x_1 \times 10^1 + x_0 \times 10^0) \times \\ (y_{n-1} \times 10^{n-1} + y_{n-2} \times 10^{n-2} + \dots + y_1 \times 10^1 + y_0 \times 10^0)$$

39

Ví dụ 2. Nhân số nguyên: Thuật toán Karatsuba (1962)

Ta có: $x = x_{n-1} x_{n-2} \dots x_1 x_0$ và $y = y_{n-1} y_{n-2} \dots y_1 y_0$

Đặt: $a = x_{n-1} x_{n-2} \dots x_{n/2+1} x_{n/2}$, Khi đó:
 $b = x_{n/2-1} x_{n/2-2} \dots x_1 x_0$, $x = a \times 10^{n/2} + b$;
 $c = y_{n-1} y_{n-2} \dots y_{n/2+1} y_{n/2}$, $y = c \times 10^{n/2} + d$,
 $d = y_{n/2-1} y_{n/2-2} \dots y_1 y_0$.

Vì thế: $z = x * y = (a \times 10^{n/2} + b) \times (c \times 10^{n/2} + d)$
 $= (a \times c) 10^n + (a \times d + b \times c) 10^{n/2} + b \times d$.

Để tính $a \times c$, $a \times d$, $b \times c$, $b \times d$ ta phải thực hiện 4 phép nhân các số có $n/2$ chữ số.

Bài toán đã cho yêu cầu thực hiện phép nhân của 2 số x và y có n chữ số: được quy về bài toán tính 4 phép nhân của số có $n/2$ chữ số

40

2.3. Nhân số nguyên: Thuật toán Karatsuba (1962)

- Neo đê qui:** Việc nhân hai số nguyên có 1 chữ số có thể thực hiện một cách trực tiếp;
 - Chia:** Nếu $n > 1$ thì tích của 2 số nguyên có n chữ số có thể biểu diễn qua 4 tích của 4 số nguyên có $n/2$ chữ số: $a*c, a*d, b*c, b*d$
 - Tổng hợp:** Để tính kết quả $z = xy$ khi đã biết 4 tích nói trên chỉ cần thực hiện các phép cộng (có thể thực hiện với thời gian $O(n)$) và phép nhân với lũy thừa của 10 (có thể thực hiện với thời gian $O(n)$, bằng việc di chuyển một số lượng số 0 thich hợp vào bên phải).
- $x = a \times 10^{n/2} + b; \quad z = x * y = (a \times 10^{n/2} + b) \times (c \times 10^{n/2} + d)$
 $y = c \times 10^{n/2} + d, \quad = (a \times c) 10^n + (a \times d + b \times c) 10^{n/2} + b \times d.$
- Đánh giá thời gian tính $T(n)$ của thuật toán:
- $T(1) = 1, n=1$ \rightarrow Có thể tăng tốc??
 $T(n) = 4T(n/2) + n, n > 1$

41

2.3. Nhân số nguyên: Thuật toán Karatsuba (1962)

Karatsuba đã phát hiện cách thực hiện việc nhân 2 số nguyên có n chữ số đòi hỏi 3 phép nhân các số có $n/2$ chữ số sau đây:

- Đặt: $U = a*c, V = b*d, W = (a+b)*(c+d)$

Khi đó: $a*d + b*c = W - U - V,$

và do đó có thể tính:

$$\begin{aligned} z &= x * y = (a \times 10^{n/2} + b) \times (c \times 10^{n/2} + d) \\ &= (a \times c) 10^n + (a \times d + b \times c) 10^{n/2} + b \times d \\ &= U \times 10^n + (W - U - V) \times 10^{n/2} + V. \end{aligned}$$

42

Thuật toán Karatsuba

```
function Karatsuba(x, y, n)
begin
    if n=1 then return x[0]*y[0]
    else
        begin
            a:= x[n-1] ... x[n/2];
            b:= x[n/2 -1] ... x[0];
            c:= y[n-1] ... y[n/2];
            d:= y[n/2-1] ... y[0];
            U:= Karatsuba(a, c, n/2);
            V:= Karatsuba(b, d, n/2);
            W:= Karatsuba(a+b, c+d, n/2);
            return U*10^n + (W-U-V)*10^{n/2} + V;
        end;
end;
```

• Thời gian tính $T(n)$ của thuật toán:
 $T(1) = 1, n=1$
 $T(n) = 3T(n/2) + cn, n > 1$
theo định lý thư: $T(n) = O(n^{\log_3 3})$
Do $\log_2 3 = 1.585 < 2 \Rightarrow$ ta có tăng tốc

Nội dung

1. Sơ đồ chung của thuật toán

2. Các ví dụ minh họa

2.1. Sắp xếp trộn (Merge sort) và sắp xếp nhanh (Quick sort)

2.2. Tìm kiếm nhị phân

2.3. Nhân số nguyên

2.4. Nhân ma trận

2.5. Lũy thừa nhị phân

2.6. Từ Fibonacci

2.4. Nhân ma trận

Cho $A = \{a_{ik}\}$ và $B = \{b_{kj}\}$ là hai ma trận kích thước $n \times n$.

Ta gọi tích của hai ma trận A và B là ma trận $C = \{c_{ij}\}$ kích thước $n \times n$, ký hiệu là $C = AB$, với các phần tử được tính theo công thức

$$c_{ij} = \sum_{k=1}^n a_{ik} b_{kj}$$

45

2.4. Nhân ma trận: thuật toán trực tiếp

procedure NaiveMatrixMultiplication()
begin

```
    for i=1 to n do
        for j=1 to n do
            begin
                c[i,j]=0;
                for k=1 to n do
                    c[i,j] = c[i,j] + a[i,k]*b[k,j];
            end;
    end;
```

$$c_{ij} = \sum_{k=1}^n a_{ik} b_{kj}$$

Thời gian tính: $O(n^3)$

46

2.4. Nhân ma trận: thuật toán chia để trị

- Chia ma trận ra thành 4 ma trận kích thước $(n/2) \times (n/2)$. Ta có

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

- trong đó $C_{11} = A_{11}B_{11} + A_{12}B_{21}$,

$$C_{12} = A_{11}B_{12} + A_{12}B_{22},$$

$$C_{21} = A_{21}B_{11} + A_{22}B_{21},$$

$$C_{22} = A_{21}B_{12} + A_{22}B_{22}.$$

8 phép nhân ma trận $(n/2) \times (n/2)$
4 phép cộng ma trận $(n/2) \times (n/2)$



- Gọi $T(n)$ là số phép nhân cần thực hiện khi tính tích của hai ma trận cấp n theo thuật toán này, ta có

$$T(n) = 8 T(n/2) + O(n^2).$$

- Từ đó theo định lý thư: $T(n) = O(n^3)$

- Không hiệu quả hơn nhân truyền thống!

47

Nhân ma trận: Công thức Strassen

Tính 7 tích ma trận:

$$P_1 = A_{11}(B_{12} - B_{22})$$

$$P_2 = (A_{11} + A_{12})B_{22}$$

$$P_3 = (A_{21} + A_{22})B_{11}$$

$$P_4 = A_{22}(B_{21} - B_{11})$$

$$P_5 = (A_{11} + A_{22})(B_{11} + B_{22})$$

$$P_6 = (A_{12} - A_{22})(B_{21} + B_{22})$$

$$P_7 = (A_{11} - A_{21})(B_{11} + B_{12})$$

Khi đó:

$$C_{11} = P_2 + P_4 + P_5 + P_6$$

$$C_{12} = P_1 + P_2$$

$$C_{21} = P_3 + P_4$$

$$C_{22} = P_1 - P_3 + P_5 - P_7$$

$\Rightarrow 7$ phép nhân và 18 phép cộng/trên các ma trận kích thước $(n/2) \times (n/2)$

Công thức Strassen: chia để trị

Thuật toán chia để trị tính tích ma trận: $C_{n \times n} = A_{n \times n} * B_{n \times n}$

- **Divide:** Chia ma trận $A_{n \times n}$ và $B_{n \times n}$ thành các ma trận con kích thước $(n/2) \times (n/2)$
- **Conquer:** Thực hiện 7 phép nhân trên các ma trận con kích thước $(n/2) \times (n/2)$ một cách đệ quy
- **Combine:** Tính ma trận kết quả $C_{n \times n}$ bằng các phép + và - trên các ma trận con kích thước $(n/2) \times (n/2)$

- Từ đó, ta có công thức đệ qui cho thời gian tính của thuật toán chia để trị phát triển trên cơ sở công thức này là

$$T(n) = 7 T(n/2) + O(n^2),$$

Theo định lý thố: $T(n) = O(n^{\log 7}) = O(n^{2.81})$

Tăng tốc!!!

Nội dung

1. Sơ đồ chung của thuật toán

2. Các ví dụ minh họa

 2.1. Sắp xếp trộn (Merge sort) và sắp xếp nhanh (Quick sort)

 2.2. Tìm kiếm nhị phân

 2.3. Nhân số nguyên

 2.4. Nhân ma trận

2.5. Lũy thừa nhị phân

 2.6. Từ Fibonacci

NGUYỄN KHÁNH PHƯƠNG 50
KHMT – SOICT- DHBK HN

2.5. Lũy thừa nhị phân

Bài toán: Tính x^n , với x, n là những số nguyên

Giả sử chúng ta không có phương thức dụng sẵn pow

Cách cài đặt đơn giản: $x^n = \underbrace{x * x * x * \dots * x}_{n \text{ lần}}$

```
int pow(int x, int n)
{
    int res = 1;
    for (int i = 0; i < n; i++) res = res * x;
    return res;
}
```

Độ phức tạp: $O(n)$

➔ Cách tăng tốc ??

NGUYỄN KHÁNH PHƯƠNG 51
KHMT – SOICT- DHBK HN

2.5. Lũy thừa nhị phân: chia để trị

Nhận xét (1):

- $x^0 = 1$
- $x^n = x \times x^{n-1}$

➔ Cần viết hàm pow:

- $\text{pow}(x, 0) = 1$
- $\text{pow}(x, n) = x \times \text{pow}(x, n - 1)$

```
int pow(int x, int n)
{
    if (n == 0) return 1;
    return x * pow(x, n - 1);
}
```

➔ Thời gian tính: $T(n) = 1 + T(n-1) \rightarrow T(n) = O(n)$

52

2.5. Lũy thừa nhị phân: chia để trị

Nhận xét (2): $x^n = x^{n/2} \times x^{n/2}$

⇒ $\text{pow}(x, n) = \text{pow}(x, n/2) \times \text{pow}(x, n/2)$

$\text{pow}(x, n/2)$ được sử dụng 2 lần, nhưng chúng ta chỉ cần tính nó một lần:

$\text{pow}(x, n) = \text{pow}(x, n/2)^2$

➔ Ta có thể sử dụng để tăng tốc trong trường hợp n là số chẵn vì khi đó $n/2$ sẽ là số nguyên

```
int pow(int x, int n)
{
    if (n == 0) return 1;
    if (n%2 != 0)
        return x * pow(x, n - 1);
    int tmp = pow(x, n/2);
    return tmp*tmp;
}
```

Thời gian tính:

$T(n) = 1 + T(n/2)$ nếu n là số chẵn

$T(n) = 1 + \frac{T(n-1)}{2}$, nếu n là số lẻ

$T(n) = 1 + [1+\frac{T(n-1)}{2}]$ vì khi n là số lẻ thì $n-1$ là số chẵn

➔ $T(n) = O(\log n) \Rightarrow$ Tăng tốc

2.5. Lũy thừa nhị phân

```
int pow(int x, int n)
{
    if (n == 0) return 1;
    if (n%2 != 0)
        return x * pow(x, n - 1);
    int tmp = pow(x, n/2);
    return tmp*tmp;
}
```

Thuật toán trên có thể áp dụng với:

- Tính x^n khi x là một số thực, khi đó $*$ là phép nhân số thực
- Tính A^n khi A là một ma trận, khi đó $*$ là phép nhân ma trận
- Tính $x^n \pmod m$ khi x là một ma trận, và $*$ là phép nhân số nguyên đồng dư m
- Tính $x*x...*x$ khi x là phần tử bất kỳ và $*$ là toán tử kết hợp bất kỳ.

Thời gian tính: $O(f^* \log n)$ với f là chi phí để thực hiện toán tử $*$

NGUYỄN KHÁNH PHƯƠNG 54
KHMT – SOICT- DHBK HN

Nội dung

1. Sơ đồ chung của thuật toán

2. Các ví dụ minh họa

2.1. Sắp xếp trộn (Merge sort) và sắp xếp nhanh (Quick sort)

2.2. Tìm kiếm nhị phân

2.3. Nhân số nguyên

2.4. Nhân ma trận

2.5. Lũy thừa nhị phân

2.6. Từ Fibonacci

NGUYỄN KHÁNH PHƯƠNG 55
KHMT – SOICT- DHBK HN

2.6. Từ Fibonacci

Dãy Fibonacci được định nghĩa đê quy như sau:

- $Fib_1 = 1$
- $Fib_2 = 1$
- $Fib_n = Fib_{n-2} + Fib_{n-1}$

Từ đó, ta thu được dãy Fibonacci: 1, 1, 2, 3, 5, 8, 13, 21...

Nếu thay điều kiện đầu $F_1 = 5$ và $F_2 = 4$ ⇒ ta có dãy 5, 4, 9, 13, 22, 35, 57,...

Điều gì xảy ra nếu điều kiện đầu không phải là số ?

Ví dụ: Điều kiện đầu là kí tự, phép + biểu thị phép nối xâu

- $g_1 = A$

- $g_2 = B$

- $g_n = g_{n-2} + g_{n-1}$

➔ Ta có dãy các xâu kí tự:

✓ A
✓ B
✓ AB
✓ BAB
✓ ABBAB
✓ BABABBABABABAB
✓ BABABBBABABABABABAB
✓

NGUYỄN KHÁNH PHƯƠNG 56
KHMT – SOICT- DHBK HN

2.6. Từ Fibonacci

Ví dụ: Điều kiện đầu là kí tự, phép + biểu thị phép nối xâu

- $g_1 = A$
 - $g_2 = B$
 - $g_n = g_{n-2} + g_{n-1}$
- ✓ A
 ✓ B
 ✓ AB
 ✓ BAB
 ✓ ABBAB
 ✓ BABABAB
 ✓ ABBABABAB
 ✓ BABABABABABABABABABAB
 ✓

Độ dài của g_n :

- $\text{length}(g_1) = 1$
 - $\text{length}(g_2) = 1$
 - $\text{length}(g_n) = \text{length}(g_{n-2}) + \text{length}(g_{n-1})$
- $\Rightarrow \text{length}(g_n) = \text{Fib}_n$

Vì vậy những xâu kí tự nhanh chóng trở nên rất dài

- $\text{length}(g_{10}) = 55$
- $\text{length}(g_{100}) = 354224848179261915075$
- $\text{length}(g_{1000}) = 434665576869374564356885276750406258025646605173717$
804024817290895365554179490518904038798400792551692
959225930803226347752096896232398733224711616429964
40906533187938298969649928516003704476137795166849228875

57

2.6. Từ Fibonacci

Ví dụ: Điều kiện đầu là kí tự, phép + biểu thị phép nối xâu

- $g_1 = A$
 - $g_2 = B$
 - $g_n = g_{n-2} + g_{n-1}$
- $\Rightarrow \text{length}(g_n) = \text{Fib}_n$
- ✓ A
 ✓ B
 ✓ AB
 ✓ BAB
 ✓ ABBAB
 ✓ BABABAB
 ✓ ABBABABAB
 ✓ BABABABABABABABABABAB
 ✓

Yêu cầu: Tim kí tự thứ i trong g_n

Ví dụ: tìm kí tự thứ i = 3 trong $g_7 = \text{AB BABBABABBAB} \rightarrow$ kí tự cần tìm là B

- Cách 1: Dễ dàng thực hiện trong thời gian $O(\text{length}(g_n)) \rightarrow$ chậm khi n lớn
- Xây dựng g_n , sau đó đưa ra kí tự thứ i của g_n .

NGUYỄN KHÁNH PHƯƠNG 58
KHMT – SOICT - DHBK HN

2.6. Từ Fibonacci

Cách 2: Chia để trị $O(n)$

- Bước 1: Không cần xây dựng g_n , chỉ cần tìm giá trị k nhỏ nhất sao cho g_k có độ dài $\geq i$.
Khi đó kí tự thứ i trong g_n cũng là kí tự thứ i trong g_k .
 - Hãy viết hàm `int findIndex(int i)` trả về giá trị k nhỏ nhất sao cho g_k có độ dài $\geq i$
 - Bước 2: Từ công thức: $g_k = g_{k-2} + g_{k-1} \rightarrow$ cần xác định xem kí tự thứ i của g_k thuộc g_{k-2} hay g_{k-1}
 - Nếu $i \leq \text{length}(g_{k-2})$ thì kí tự thứ i thuộc g_{k-2}
 - Vậy kí tự thứ i của g_k là kí tự thứ i của g_{k-2} . Lặp lại bước 2 với $k = k-2$
 - else kí tự thứ i thuộc g_{k-1}
 - Vậy kí tự thứ i của g_k là kí tự thứ $(i - \text{length}(g_{k-2}))$ của g_{k-1} . Lặp lại bước 2 với $k = k-1$
- Lặp lại bước 2 với giá trị k giảm dần cho đến khi $k \leq 2$

Hãy viết hàm `int findLengthTerm(int k)` trả về độ dài của g_k

Hãy viết hàm `char Fibonacci(int n, int i)` dùng hai hàm trên để giải bài toán đã cho

59

Bài tập về nhà: Fibonacci word

For any two strings of digits, A and B , we define $F_{A,B}$ to be the sequence $(A, B, AB, BAB, ABBAB, \dots)$ in which each term is the concatenation of the previous two.

Further, we define $D_{A,B}(n)$ to be the n -th digit in the first term of $F_{A,B}$ that contains at least n digits.

Example:

Let $A = 1415926535$, $B = 8979323846$. We wish to find $D_{A,B}(35)$, say.

The first few terms of $F_{A,B}$ are:

- 1415926535
- 8979323846
- 14159265358979323846
- 897932384614159265358979323846
- 14159265358979323846897932384614159265358979323846

Then $D_{A,B}(35)$ is the 35-th digit in the fifth term, which is 9.

You are given q triples (A, B, n) . For all of them find $D_{A,B}(n)$.

60

Bài tập về nhà: Fibonacci word

Input Format

First line of each test file contains a single integer q that is the number of triples. Then q lines follow, each containing two strings of decimal digits a and b and positive integer n .

Constraints

Output Format

- $1 \leq q \leq 100$
- Print exactly q lines with a single decimal digit on each: value of $D_{A,B}(n)$ for the corresponding triple.
- $1 \leq \text{length}(a), \text{length}(b) \leq 100$
- $1 \leq n \leq 2^{100}$

Sample Input 0

```
2
1415926535 8979323846 35
14159265358979323846264338327950288419716939937510582097494459230781640620620899862803
48253421170679
82148866513282306647093844669550582231725359408128481117450284102701938521105559644622
94895493038196 104683731294243150
```

Sample Output 0

```
9
8
```

61