14. Độ phức tạp của thuật toán

Nói về độ phức tạp của thuật toán thì ta quan tâm tới 2 yếu tố: độ lớn của kiểu dữ liệu và thời gian chạy chương trình. Với cấu hình máy tính hiện nay, thì độ phức tạp của thuật toán ta thường quan tâm đến mặt thời gian hơn, bởi vì nếu một thuật toán cho ra một kết quả đúng nhưng thời gian thực hiện thuật toán cần một khoảng thời gian lớn đến nỗi không thể chấp nhận được (Ví dụ bài toán tháp Hà Nội nếu dùng đệ quy để tính toán thì với tốc độ máy tính hiện nay phải giải nó khoảng vài ngàn năm). Do đó yêu cầu thời gian được đưa ra đầu tiên. Thuật toán được gọi là hay thì phải có thời gian thực hiện ngắn và tiết kiệm tài nguyên máy tính. Sự hao phí tài nguyên máy tính liên quan đến phần cứng máy tính. Vì vậy mà những thuật toán đưa ra thường lấy thời gian để tính độ phức tạp hơn là tài nguyên (vì mỗi máy khác tài nguyên). Vì vậy, độ phức tạp của thuật toán là được hiểu nôm na là thời gian thực hiện của thuật toán. Ký hiệu độ phức tạp của thuật toán là *O* lớn.

Ta có tính chất $O(f(x)+g(x))=max\ (O(f(x)), O(g(x)))$

Ở đây chúng ta hiểu nôm na là đánh giá về mặt thời gian. Và cách hiểu thông thường nhất chính là số lần lặp tối đa của một đoạn nào đó trong chương trình.

Ví dụ 1. Xét câu lệnh: $x = x+1;$

Độ phức tạp của câu lệnh trên là $O(1)$. Vì đây là câu lệnh đơn nó được thực hiện *1* lần.

Ví dụ 2. Xét câu lệnh sau:

```
for (int i=1; i<=10; i++) x = x+ 1;
```

Câu lệnh trên được lặp *10* lần, mỗi lần lặp thì nó thực hiện một lệnh đơn $x = x+1$ suy ra câu lệnh trên có độ phức tạp là $O(10)$.

Ví dụ 3. Xét câu lệnh sau:

```
for(int i=1; i<=n; i++) x = x+ 1;
```

Câu lệnh trên được lặp *n* lần, nên độ phức tạp của thuật toán là $O(n)$.

Ví dụ 4. Xét câu lệnh sau:

```
for(int i=1; i<= m; i++)
    for(int j=1; j<= n; j++)
        x = x+ a;
```

Đây là 2 câu lệnh *for* lồng nhau, nên nó lặp $m*n$ lần, suy ra độ phức tạp của thuật toán là $O(m*n)$.

Ví dụ 5. Xét đoạn chương trình sau:

```
x = 1; //O(1)
for (int i =1; i <= n; i++) x = x+i;        //O(n)
for (int i =1; i <= n; i++)
        for (int j=1; j <=n; j++) x = x+1; //O(n^2)
```

Độ phức tạp của đoạn chương trình trên là $max(O(1), O(n), O(n^2)) = O(n^2)$.

Ví dụ 6. Xét đoạn chương trình tìm kiếm nhị phân sau:

```
dau =1; cuoi = n;
while (dau <= cuoi) {
        giua = (dau+cuoi)/2;
        if (a[giua] = x) {
                cout << "tim thay";
                break;
        }
        else    if (a[giua] > x)  cuoi = giua -1;


                        else dau = giua +1;
}
```

Câu lệnh *dau =1; cuoi = n;* là câu lệnh đơn có độ phức tạp là $O(1)$. Còn cả khối lệnh phía sau là đoạn tìm kiếm nhị phân trên mảng có n phần tử *a[1], a[2], a[3],... a[n]*. Nó cứ chia đôi ra để tìm. Suy ra số lần tối đa của việc chia đôi ra chính là độ phức tạp của thuật toán. Giả sử số lần chia đôi là k, thì ta dễ dàng thấy $2^k = n$. Từ đây là suy ra $k = log_2 n$. Độ phức tạp của đoạn chương trình trên là $O(log_2 n)$ hay nói một cách ngắn gọn là $O(logn)$.

Ví dụ 7. Xét đoạn chương trình sắp xếp sau:

```
for (int i= 1; i <= n-1; i++)
        for (int j = i+1; j <= n; j++)
                if (a[i]>a[j]){
                        tg = a[i];
                        a[i] = a[j];
                        a[j] = tg;
                }
```

Đoạn chương trình trên có cấu trúc *for* lồng nhau, còn sau lệnh *for* là các câu lệnh đơn

- Khi *i =1* thì nó lặp *n-1* lần;
- Khi *i =2* thì nó lặp *n-2* lần;
- Khi *i =3* thì nó lặp *n-3* lần;
- .....
- Khi *i = n-1* thì nó lặp *1* lần;

Vậy số lần lặp tối đa của đoạn chương trình trên là: *(n-1) + (n-2) + (n-3) + ... + 2 + 1*. Đây là tổng các số tự nhiên từ *1* đến *n-1* có kết quả là *(n*(n-1))/2*. Vậy độ phức tạp của thuật toán là *O(n*(n-1)/2)*. Nhiều lúc chúng ta không cần đánh giá chính xác mà chỉ cần lấy xấp xỉ. Nên ta cũng có thể nói độ phức tạp trên bằng $O(n^2)$.

Từ đây ta có thể dễ dàng đánh giá độ phức tạp của một số thuật toán thông dụng như: tìm ước chung lớn nhất của *2* số *a* và *b*: *max(O(loga), O(logb))*; tìm kiếm nhị phân: *O(logn)*; kiểm tra số nguyên tố thông thường: $O(\sqrt{n})$; sàng nguyên tố: *O(nlogn)*; hàm *sort* trong thư viện: *O(nlogn)*...; Nhiều lúc đánh giá trực tiếp về độ phức tạp là khó khăn: ví dụ như đệ quy sinh nhị phân

độ dài *n*, rõ ràng ta thấy nếu độ dài *n* thì ta đệ quy sinh ra được 2^n dãy nhị phân. Vậy độ phức tạp là $O(2^n)$, vì nó lặp 2^n để sinh ra số nhị phân.

Trong khi thi, thông thường mỗi test có thời gian chấm là *1 giây* (*1s*). Trong *1s* thông thường máy tính làm được khoảng 4.10^8 phép tính. Chính vì vậy khi đọc giới hạn của đề bài thì ta sẽ biết làm bài đó với độ phức tạp bao nhiêu để được điểm tối đa. Đánh giá độ phức tạp thuật toán là cực kỳ quan trọng trong thi cử. Nếu đánh giá sai độ phức tạp thuật toán dẫn đến bài của mình làm thì mình không tự chấm được chính xác bao nhiêu điểm (nếu như code đúng). Ví dụ: nếu cách làm là *O(n)* thì bài đó mình có thể làm được với $n \leq 4.10^8$; Nếu là *O(nlogn)* thì ta tính $nlogn = 4.10^8$, suy ra được $n \leq 10^7$; Nếu là $O(n^2)$ thì tương tự ta tính được bài đó làm được với $n \leq 2.10^4$...

15. Sử dụng hàm sort trong thư viện

Giả sử ta có mảng a gồm có *n* phần tử nguyên hoặc thực hoặc kí tự hoặc kiểu xâu, kiểu bản ghi. Trong thư viện của C++ có hàm *sort* dùng để sắp xếp mảng, độ phức tạp của nó là *O(nlogn)* tương đương với thuật toán sắp xếp *quicksort*;

Cách 1:

sort (a+1, a+n +1); //nghĩa là sắp xếp mảng a thành mảng không giảm (hay gọi là tăng không ngặt) từ vị trí 1 đến vị trí *n* (*a[1]* đến *a[n]*)

sort (a, a+n); //nghĩa là sắp xếp mảng *a* thành mảng không giảm từ phần tử thứ 0 đến phần tử thứ *n-1* (*a[0]* đến *a[n-1]*)

Tổng quát có thể sắp xếp không giảm từ vị trí *i* đến vị trí *j*: *sort (a+i, a+j+1);*

Nếu là mảng các phần tử là kiểu số thì ta đổi dấu tất cả các phần tử, rồi sắp xếp tăng như hàm ở trên, rồi lại đổi dấu lại thì ta được mảng được sắp xếp giảm. Hoặc sau khi sắp xếp tăng, ta duyệt mảng từ phần tử cuối về phần tử đầu → duyệt dãy giảm.

Cách 2: Cách tự viết điều kiện

// sắp xếp theo điều kiện hàm cmp từ phần tử a[1] đến a[n]

```
sort (a+1, a+n+1, cmp);
```

// sắp xếp theo điều kiện hàm cmp từ phần tử a[0] đến a[n-1]

```
sort (a, a+n, cmp);
```

Cách viết hàm *cmp* như sau: hàm *cmp* này phải viết ở trước khi gọi hàm *sort*. *cmp* chỉ là tên do người lập trình đặt, ta có thể đặt tên khác cũng không sao. Ví dụ sosanh thì phải viết hàm có tên là sosanh...

Cách viết hàm *cmp*

```
// Sắp xếp tăng
bool   cmp (int a, int b){
    return a<b;
}
```

```
// Sắp xếp giảm
bool   cmp (int a, int b){
    return a>b;
}
```

Đối với đối số *a, b* trong hàm *cmp* ta cứ hiểu nôm na là *a* đứng trước *b* thì khi mảng sắp xếp giảm thì trả kết quả là true nếu $a > b$ (rõ ràng $a > b$ thì mới có dãy giảm vì *a* đứng trước *b* trong mảng). Còn sắp xếp tăng thì trả kết quả *true* nếu $a < b$ (rõ ràng $a < b$ thì mới có dãy giảm vì *a* đứng trước *b* trong mảng). Ta chú ý thêm 1 điều: mảng có kiểu dữ liệu gì thì đối số trong hàm *cmp* phải có kiểu dữ liệu y như vậy. Ví dụ mảng có các phần tử là *long long* thì các đối số *a và b* của hàm *cmp* cũng có phần tử kiểu *long long*. Nếu mảng *a* là kiểu dữ liệu là bản ghi có tên là *hocsinh* thì đối số của hàm *cmp* cũng phải kiểu dữ liệu *hocsinh*. Ví dụ mảng *a* là kiểu bản ghi có tên là *hocsinh* gồm có trường *ten, ngaysinh, diem*. Yêu cầu hãy sắp xếp theo *diem* giảm dần. Thì ta viết *cmp* như sau:

```
bool   cmp (hocsinh  a, hocsinh  b){
    return a.diem > b.diem;
}
```

Hoặc để dễ hiểu hơn nữa ta viết tường tận như sau:

```
bool   cmp (hocsinh  a, hocsinh  b){
    if( a.diem > b.diem) return true;
    return  false;
}
```

Ví dụ. Hãy nhập N học sinh có các thuộc tính như trên. Sắp xếp theo điểm tăng dần, nếu 2 điểm bằng nhau thì sắp xếp theo tên giảm dần.

```
#include <bits/stdc++.h>
#define  N 1001
using namespace std;
```

```
struct  hocsinh{
  string  ten;
  string  ngaysinh;
  double  diem;
};
hocsinh  a[N];
int i, n;
bool  cmp (hocsinh  a, hocsinh  b){
  if (a.diem < b.diem) return true;
  if (a.diem = b.diem)
    if (a.ten > b.ten)  return true;
  return false;
}
int main(){
  cin >> n;
  for (int i =1 ; i<= n; i++) {
    cin >> a[i].ten;
    cin >> a[i].ngaysinh;
    cin >> a[i].diem;
  }
  sort(a+1, a+n+1, cmp);
  // in ra danh sach mỗi học sinh trên 1 hàng
  for (int i = 1; i <= n; i++ )
    cout << a[i].ten << " "  << a[i].ngaysinh << " "<< a[i].diem << endl;
  return 0;
}
```