Chương 1: TỔNG QUAN VỀ CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

Nội dung

- Vai trò của cấu trúc dữ liệu trong một đề án tin học
- Các tiêu chuẩn đánh giá cấu trúc dữ liệu
- Kiểu dữ liệu
- Đánh giá độ phức tạp của giải thuật

1.1. Vai trò của cấu trúc dữ liệu trong một đề án tin học

1.1.1. Xây dựng cấu trúc dữ liệu

- Các thành phần dữ liệu thực tế đa dạng, phong phú, quan hệ với nhau=> cần phải tổ chức , xây dựng các cấu trúc thích hợp nhất để:
  - Phản ánh chính xác các dữ liệu thực tế
  - Dễ dàng dùng máy tính để xử lý

1.1. Vai trò của cấu trúc dữ liệu trong một đề án tin học

1.1.1. Xây dựng cấu trúc dữ liệu

- Dữ liệu có thể là:
  - Input data
  - Trung gian
  - Output

1.1. Vai trò của cấu trúc dữ liệu trong một đề án tin học

1.1.2. Xây dựng giải thuật

- Giải thuật hay còn gọi là thuật toán là phương pháp hay cách thức (method) để giải quyết vấn đề.
- Cách mô tả giải thuật:
  - Ngôn ngữ tự nhiên (natural language)
  - Sơ đồ (flow chart)
  - Mã giả (pseudo code) thường được sử dụng
- Ví dụ: Mô tả giải thuật giải phương trình
  - $ax+b=0$
  - $ax^2+bx+c=0$

1.1. Vai trò của cấu trúc dữ liệu trong một đề án tin học

Mã giả

Khai báo thuật toán

Thuật toán <tên TT> (<tham số>)

Input: <dữ liệu vào>

Output: <dữ liệu ra>

<Các câu lệnh>

End <tên TT >

1.1. Vai trò của cấu trúc dữ liệu trong một đề án tin học

VD: Giải thuật giải PT ax+b=0

Thuật toán Giai_PT

```
Input: hệ số a,b
Output: nghiệm của PT
Begin
        Nhập vào các hệ số a,b
        if (a ≠ 0) PT có nghiệm x=-b/a
        else
            if (b ≠ 0) PT vô nghiệm
            else PT vô số nghiệm
 End Giai_PT
```

1.1. Vai trò của cấu trúc dữ liệu trong một đề án tin học

1.1.3. Mối liên hệ giữa cấu trúc dữ liệu và giải thuật

Cấu trúc dữ liệu + Giải thuật = Chương trình

1.1. Vai trò của cấu trúc dữ liệu trong một đề án tin học

1.2. Các tiêu chuẩn đánh giá cấu trúc dữ liệu

- Một cấu trúc dữ liệu tốt phải thỏa mãn các tiêu chuẩn sau :
  - Phản ánh đúng thực tế
  - Phù hợp với các thao tác trên đó
  - Tiết kiệm tài nguyên hệ thống

1.3. Kiểu dữ liệu

1.3.1. Định nghĩa kiểu dữ liệu

Kiểu dữ liệu T được xác định bởi một bộ <V,O>

- V : tập các giá trị hợp lệ mà một đối tượng kiểu T có thể lưu trữ
- O : tập các thao tác xử lý có thể thi hành trên đối tượng kiểu T.

- Ví du:

Giả sử có kiểu dữ liệu **mẫu tự** = <Vc ,Oc> với

Vc = { a-z,A-Z}

Oc = { lấy mã ASCII của ký tự, biến đổi ký tự thường thành ký tự hoa …}

Giả sử có kiểu dữ liệu **số nguyên** = <Vi ,Oi> với

Vi = { -32768..32767}

Oi = { +, -, *, /, %}

1.3. Kiểu dữ liệu

1.3.1. Định nghĩa kiểu dữ liệu

Các thuộc tính của 1 KDL bao gồm:

- Tên KDL
- Miền giá trị
- Kích thước lưu trữ
- Tập các toán tử tác động lên KDL

1.3. Kiểu dữ liệu

1.3.2. Các kiểu dữ liệu cơ bản

- Kiểu có thứ tự rời rạc: số nguyên, ký tự, logic , liệt kê, miền con
- Kiểu không rời rạc: số thực

1.3.3. Các kiểu dữ liệu có cấu trúc

- Kiểu chuỗi ký tự
- Kiểu mảng
- Kiểu mẫu tin (cấu trúc)
- Kiểu union

1.3. Kiểu dữ liệu

Ví dụ: Để mô tả một đối tượng sinh viên, cần quan tâm đến các thông tin sau:

- Mã sinh viên: chuỗi ký tự
- Tên sinh viên: chuỗi ký tự
- Ngày sinh: kiểu ngày tháng
- Nơi sinh: chuỗi ký tự
- Điểm thi: số thực

1.3. Kiểu dữ liệu

Các kiểu dữ liệu cơ sở cho phép mô tả một số thông tin như :

- float Diemthi;

Các thông tin khác đòi hỏi phải sử dụng các kiểu có cấu trúc như :

- char masv[15];
- char tensv[15];
- char noisinh[15];

1.3. Kiểu dữ liệu

Để thể hiện thông tin về ngày tháng năm sinh cần phải xây dựng một kiểu bản ghi,

```
typedef struct tagDate
{
    char ngay;
    char thang;
    char thang;
} Date;
```

1.3. Kiểu dữ liệu

- kiểu dữ liệu thể hiện thông tin về một sinh viên :

```
typedef struct tagSinhVien
        {
         char masv[15];
         char tensv[15];
         char noisinh[15];
         Date ngaysinh;
         int Diem thi;
       } SinhVien;
```

1.4. Đánh giá độ phức tạp của giải thuật

1.4.1 Các bước phân tích thuật toán

- Bước đầu tiên: xác định đặc trưng dữ liệu sẽ được dùng làm dữ liệu nhập của thuật toán và quyết định phân tích nào là thích hợp.
- Bước thứ hai: nhận ra các thao tác trừu tượng của thuật toán để tách biệt sự phân tích với sự cài đặt.
- Bước thứ ba: phân tích về mặt toán học, với mục đích tìm ra các giá trị trung bình và trường hợp xấu nhất cho mỗi đại lượng cơ bản.

1.4. Đánh giá độ phức tạp của giải thuật

1.4.2 Sự phân lớp các thuật toán

- Hằng số: : Hầu hết các lệnh của các chương trình đều được thực hiện 1 lần hay nhiều nhất chỉ một vài lần => thời gian chạy của nó là hằng số.
- logN: Khi thời gian chạy của chương trình là logarit tức là thời gian chạy chương trình tiến chậm khi N lớn dần.
- N: Khi thời gian chạy của một chương trình là tuyến tính.

1.4. Đánh giá độ phức tạp của giải thuật

1.4.2 Sự phân lớp các thuật toán

- NlogN: thời gian chạy tăng dần lên cho các thuật toán mà giải một bài toán bằng cách
  - Tách nó thành các bài toán con nhỏ hơn
  - Giải quyết chúng một cách độc lập
  - Tổ hợp các lời giải
- N^2: Khi thời gian chạy của một thuật toán là bậc hai, trường hợp này chỉ có ý nghĩa thực tế cho các bài toán tương đối nhỏ.
- N^3: một thuật toán xử lý các bộ ba của các phần tử dữ liệu (chẳng hạn là ba vòng lặp lồng nhau) có thời gian chạy bậc ba và cũng chỉ có ý nghĩa thực tế trong các bài toán nhỏ

1.4. Đánh giá độ phức tạp của giải thuật

1.4.2 Sự phân lớp các thuật toán

- 2^N: một số ít thuật toán có thời gian chạy lũy thừa lại thích hợp trong một số trường hợp thực tế.

1.4.3 Phân tích trường hợp trung bình

- Trường hợp xấu nhất
- Trường hợp tốt nhất
- Trường hợp trung bình

QUY TẮC XÁC ĐỊNH ĐỘ PHỨC TẠP

- Độ phức tạp tính toán của giải thuật: O(f(n))
- Việc xác định độ phức tạp tính toán của giải thuật trong thực tế có thể tính bằng một số quy tắc đơn giản sau:
  - **Quy tắc bỏ hằng số**:

T(n) = O(c.f(n)) = O(f(n)) với c là một hằng số dương

  - **Quy tắc lấy max**:

T(n) = O(f(n)+ g(n)) = O(max(f(n), g(n)))

  - **Quy tắc cộng**:

T1(n) = O(f(n)) T2(n) = O(g(n))

T1(n) + T2(n) = O(f(n) + g(n))

  - **Quy tắc nhân**:

Đoạn chương trình có thời gian thực hiện T(n)=O(f(n))

Nếu thực hiện k(n) lần đoạn chương trình với k(n) = O(g(n)) thì độ phức tạp sẽ là O(g(n).f(n))

Trong một thuật toán, ta chú ý đặc biệt đến một phép toán gọi là phép toán tích cực. Đó là phép toán mà số lần thực hiện không ít hơn các phép toán khác

- Ví dụ 1:

```
s = 0;
for (i=0; i<=n;i++){
        p =1;
        for (j=1;j<=i;j++)
                p=p * x / j;
        s = s+p;
}
```

- Phép toán tích cực là p = p * x / j
- Số lần thực hiện phép toán này là
  - $0+1+2+\ldots+n = n(n-1)/2 = n^2/2 - n/2$
- => Vậy độ phức tạp là $O(n^2/2 - n/2)$
- $= O(n^2/2)$ sử dụng quy tắc lấy max
- $= O(n^2)$ sử dụng quy tắc bỏ hằng số

Ví dụ 2:


```
        s = 1; p = 1;
for (i=1;i<=n;i++) {
            p = p * x / i;
            s = s + p;
}
```

Phép toán tích cực là p = p * x / i

Số lần thực hiện phép toán là n

=> Vậy độ phức tạp của thuật toán là O(n)

Ví dụ 3:

s=n*(n-1) /2;
=> Độ phức tạp của thuật toán là O(1), nghĩa là thời gian tính toán không phụ thuộc vào n

Ví dụ 4:

Thuật toán tính tổng các số từ 1 đến n

```
s=0;
for (i= 1;i<=n;i++)
        s=s+i;
```

Vòng lặp lặp n lần phép gán s = s+i, nên thời gian tính toán tỉ lệ thuận với n, tức độ phức tạp là O(n).

=> độ phức tạp là O(n)

Ví dụ 5:


```
for (i= 1;i<=n;i++)
        for (j= 1;j<=n;j++)
                //Lệnh
```

=> Dùng quy tắc nhân ta có O(n^2)

tương tự như vậy ta sẽ có O(n^3), O(n^4).

Ví dụ 6:


```
for (i= 1;i<=n;i++)
        for (j= 1;j<=m;j++)
                //Lệnh
```

=> Dùng quy tắc nhân ta có O(n*m)

Ví dụ 7:


```
for (i= 1;i<=n;i++)
        //lệnh1
for (j= 1;j<=m;j++)
        //lệnh 2
```

=> Dùng quy tắc cộng và quy tắc lấy max, sẽ có

O(max (n,m))

Ví dụ 8:


```
for (i= 1;i<=n;i++) {
        for (u= 1;u<=m;u++)
                for (v= 1;v<=n;v++)
                        //lệnh
        for j:= 1 to x do
                for k:= 1 to z do
                        //lệnh
}
```

=> O(n*max (n*m,x*z))

Ví dụ 9:


```
for (i= 1;i<=n;i++)
        for (j= 1;j<=m;j++) {
                for (k= 1;k<=x;k++)
                        //lệnh
                for (h= 1;h<=y;h++)
                        //lệnh
        }
```

=> O(n*m* max (x,y))

Ví dụ 10:


```
for (i= 1;i<=n;i++)
          for (u= 1;u<= m;u++)
                    for (v= 1;v<=n;v++)
                              //lệnh
for (j= 1;j<=x;j++)
                    for (k= 1;k<=z;k++)
                              //lệnh
=> O(max (m*n^2,x*z))
```

Ví dụ 11: Đoạn chương trình tính tổng 2 đa thức

$P(x) = x_m x^m + a_{m-1} x^{m-1} + \ldots + a_1 x + a_0$

$Q(x) = b_n x^n + b_{n-1} x^{n-1} + \ldots + b_1 x + b_0$

```
if (m<n) p = m; else p =n;
for (i=0;i<=p;i++)
        c[i]=a[i] + b[i];
if (p<m)
        for (i=p+1;i<=m;i++) c[i] = a[i];
else
        for (i=p+1;i<=n;i++) c[i] = b[i];
while (p>0 && c[p] = 0) p = p-1;
```

=> Độ phức tạp: O(max(m,n))

Ví dụ 12: Đoạn chương trình tính tích hai đa thức

$P(x) = x_m x^m + a_{m-1} x^{m-1} + \ldots + a_1 x + a_0$

$Q(x) = b_n x^n + b_{n-1} x^{n-1} + \ldots + b_1 x + b_0$

```
p = m+n;
for (i=0;i<=p;i++) c[i] = 0;
for (i=0;i<=m;i++)
        for (j=0;j<=n;j++)
                c[i+j] = c[i+j] + a[i] + b[j];
```

=> Độ phức tạp là O(m.n)

Ví dụ 13:

$x^2 = O(x^5)$

$\sin(x) = O(x)$

$1/x = O(1)$

Xét hàm số $f(x) = x^2+2x+3$.

Rõ ràng $f(x) = O(x^2)$

$kx^2 = O(x^3)$ với $k>0$

$1/(1+x^2) = O(1)$

$\sin(x) = O(1)$

Ví dụ 14:

Cho $f(x) = a_0 + a_1 x^1 + a_2 x^2 + ... + a_{n-1}x^{n-1} + a_n x^n$, trong đó ai, i=0,1,...n, là các số thực.
Khi đó $f(x) = O(x^n)$.

Ví dụ 15:

Đánh giá hàm giai thừa f(n) = n!

Quy ước:

0! = 1

1! =1;

3! = 1.2.3 = 6;

5! = 120;

10 ! = 362,880;

11! = 39,916,800;

20! = 2,432,902,008,176,640,000

Rõ ràng $n! < n^n$.

Điều này chứng tỏ $n! = O(n^n)$.

Từ đó suy ra Log(n!) = O(n log n).

Ví dụ 16:

Tìm đánh giá hàm f(n)= n log (n!) + ($3n^2$ +2n)logn.

Theo các ví dụ đã nêu ở trên ta có log(n!) = O(nlogn), từ đó dễ dàng suy ra rằng n log (n!) = O(n^2logn);
Mặt khác ta có $3n^2$ +2n =O(n^2),
hay suy ra ($3n^2$ +2n) log n = O(n^2 log n).
Cuối cùng ta có: f(n) = O(n^2 log n)

Ví dụ 17:

Tìm đánh giá đối với hàm $f(n) = (n+3) \log (n^2+4) + 5n^2$.

Tìm đánh giá tốt nhất của hàm $f(x) = 2^x + 23$.

Ví dụ 18:

Đánh giá số phép chia số nguyên của thuật toán Euclid để tìm ước số chung lớn nhất của hai số nguyên a và b,
a>b.

```
x:= a; y:=b;
While y>0
        r := x mod y;
        x := y;
        y := r;
End While
```

Đáp số: $O(\log_2 b)$.
Định lý Lamé: Cho a và b là các số nguyên dương với a >= b. Số phép chia cần thiết để tìm USCLN(a,b) nhỏ hơn hoặc bằng 5 lần số chữ số của b trong hệ thập phân (hay nói cách khác thuộc $O(\log_2 b)$).