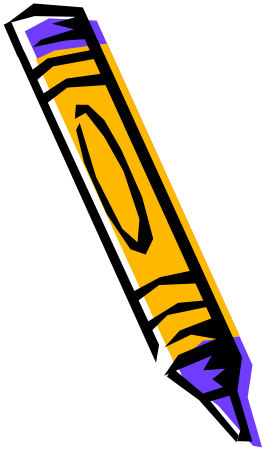
CHƯƠNG I



GIỚI THIỆU CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

Giảng viên: Trần Thị Kim Chi

NỘI DUNG

1. Khái niệm về CTDL và thuật toán
2. Đánh giá cấu trúc dữ liệu và giải thuật
3. Độ phức tạp của thuật toán
4. Quy tắc đánh giá độ phức tạp của thuật toán
5. Các ví dụ về độ phức tạp của thuật toán
6. Ôn tập ngôn ngữ Python
7. Câu hỏi và bài tập

https://www.geeksforgeeks.org/python-oops-concepts/?ref=lbp

1 KHÁI NIỆM VỀ CTDL & GT

- Dữ liệu là dữ liệu như văn bản, hình ảnh, âm thanh, video,…mà chưa có ý nghĩa rõ ràng, chưa được xử lý được đưa vào (input data) và lưu trữ trên máy tính. Dữ liệu có thể là dữ liệu vào (input data), dữ liệu trung gian hoặc dữ liệu đưa ra (output data).
- Cấu trúc dữ liệu là sự sắp xếp có logic của thành phần dữ liệu được kết hợp với nhau và là tập hợp các thao tác chúng ta cần để truy xuất các thành phần dữ liệu.
- Ví dụ: thư viện
  - Bao gồm các sách
  - Truy cập/tìm kiếm một cuốn sách nào đó đòi hỏi phải biết cách sắp xếp của các sách
  - Người dùng truy cập sách chỉ thông qua người quản lý thư viện.

2 ĐÁNH GIÁ CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

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}

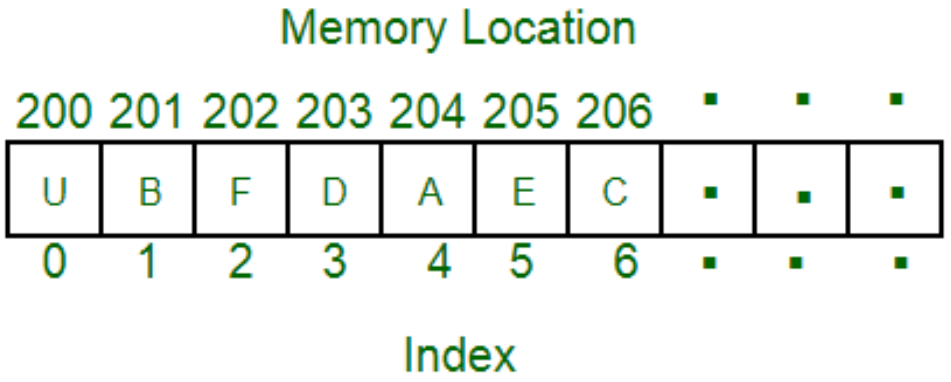
Oi = { +, -, *, /, %}

2 ĐÁNH GIÁ CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

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



2 ĐÁNH GIÁ CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

Các kiểu dữ liệu cơ bản

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

Các kiểu dữ liệu có cấu trúc

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

Ví dụ: Để mô tả một đối tượng sinh viên, 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

2 ĐÁNH GIÁ CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

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];

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;
```

2 ĐÁNH GIÁ CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

Kiểu dữ liệu trừu tượng (Abstract Data Type)

Là mô hình toán học và những phép toán thực hiện trên mô hình toán học này.

Ví dụ:ADT List

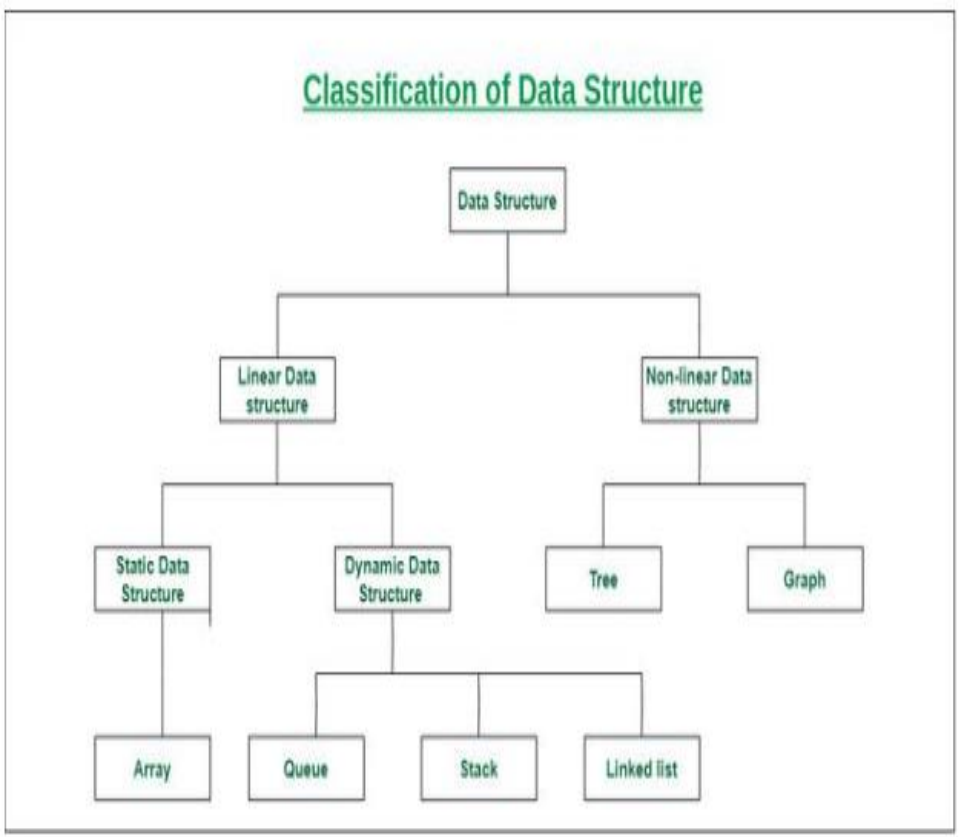
Dữ liệu: Các nút

Các phép toán:

- Thêm 1 nút mới
- Loại bỏ 1 nút
- Tìm kiếm 1 nút có giá trị cho trước.

2 ĐÁNH GIÁ CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

- **Cấu trúc dữ liệu tuyến tính:** các phần tử dữ liệu được sắp xếp tuần tự hoặc tuyến tính, trong đó mỗi phần tử được gắn với các phần tử liền kề trước và tiếp theo của nó, được gọi là cấu trúc dữ liệu tuyến tính.
  - **Cấu trúc dữ liệu tĩnh:** có kích thước bộ nhớ cố định. Việc truy cập các phần tử trong cấu trúc dữ liệu tĩnh dễ dàng hơn.
  - **Cấu trúc dữ liệu động:** kích thước không cố định. Nó có thể được cập nhật ngẫu nhiên trong thời gian chạy.



Cấu trúc dữ liệu phi tuyến tính: các phần tử dữ liệu không được đặt tuần tự hoặc tuyến tính được gọi là cấu trúc dữ liệu phi tuyến tính. Trong cấu trúc dữ liệu phi tuyến tính, chúng ta không thể duyệt tất cả các phần tử chỉ trong một lần

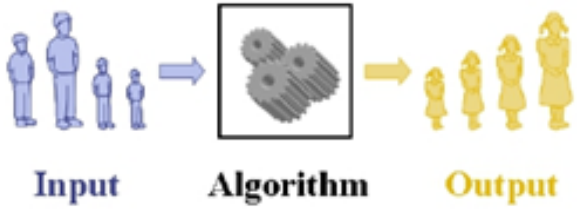
2 ĐÁNH GIÁ CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

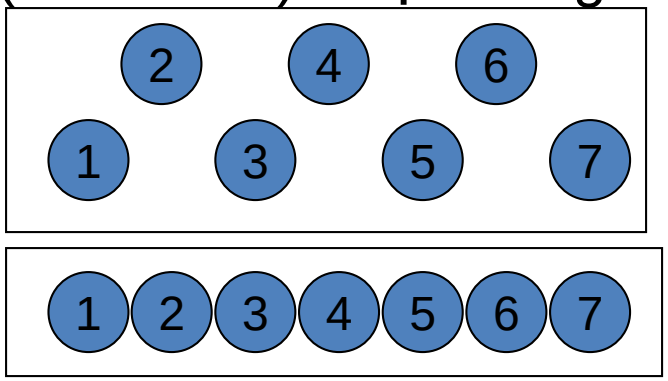
Các tiêu chuẩn đánh giá cấu trúc dữ liệu

- **Phản ánh đúng thực tế**: Cần xem xét kỹ lưỡng cũng như dự trù các trạng thái biến đổi của dữ liệu trong chu trình sống để có thể chọn CTDL lưu trữ thể hiện chính xác đối tượng thực tế.
- **Phù hợp với các thao tác trên đó**: phải dễ dàng trong việc thao tác dữ liệu. Tăng tính hiệu quả của đề án, việc phát triển các thuật toán đơn giản, tự nhiên hơn => chương trình đạt hiệu quả cao hơn về tốc độ xử lý.
- **Tiết kiệm tài nguyên hệ thống:** CTDL chỉ nên sử dụng tài nguyên hệ thống vừa đủ để đảm nhiệm được chức năng của nó. Loại tài nguyên cần quan tâm là : CPU và bộ nhớ.

2 ĐÁNH GIÁ CTDL & GT

- **Giải thuật (Algorithm):**
  - Còn gọi là thuật toán là tập các bước *có thể tính toán được* để đạt được kết quả mong muốn. (A *computable* set of steps to achieve a desired result)
  - *Một thuật toán là một bản liệt kê các chỉ dẫn, các quy tắc cần thực hiện theo từng bước xác định nhằm giải quyết một bài toán đã cho trong một khoảng thời gian hữu hạn.*
  - Giải thuật được xây dựng trên cơ sở của cấu trúc dữ liệu đã được chọn.
  - Giải thuật có thể được minh họa bằng ngôn ngữ tự nhiên (natural language), bằng sơ đồ (flow chart) hoặc bằng mã giả (pseudo code).
  - Ví dụ: Sắp xếp các phần tử





2 ĐÁNH GIÁ CTDL & GT

Khai báo Giải thuật (Algorithm):

Thuật toán <tên TT> (<tham số>)
Input: <dữ liệu vào>
Output: <dữ liệu ra>
<Các câu lệnh>
End <tên TT >

2 ĐÁNH GIÁ CTDL & GT

- **Biểu diễn Giải thuật (Algorithm) bằng ngôn ngữ tự nhiên:**
- Ví dụ: Tính tổng các số nguyên lẻ từ 1→n

  Input: n

  Output: Tổng

  – B1: S=0
  – B2: i=1
  – B3: Nếu i>n thì sang B7, ngược lại sang B4
  – B4: S=S+i
  – B5: i=i+2
  – B6: Quay lại B3
  – B7: Tổng cần tìm là S

- **Biểu diễn Giải thuật (Algorithm) bằng mã giả:**

Ví dụ: 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
```

2 ĐÁNH GIÁ CTDL & GT

Các tính chất quan trọng của giải thuật là:

- **Hữu hạn (finiteness):** giải thuật phải luôn luôn kết thúc sau một số hữu hạn bước.
- **Xác định (definiteness):** mỗi bước của giải thuật phải được xác định rõ ràng và phải được thực hiện chính xác, nhất quán.
- **Hiệu quả (effectiveness):** các thao tác trong giải thuật phải được thực hiện trong một lượng thời gian hữu hạn.
- **Tính đúng đắn:** Thuật toán phải cho kết quả đúng theo yêu cầu của bài toán đặt ra.
- **Tính tổng quát:** Thuật toán phải áp dụng được cho mọi bài toán cùng loại, với mọi dữ liệu đầu vào như đã được mô tả.

2 TẦM QUAN TRỌNG CTDL & GT

Tầm quan trọng của CTDL & giải thuật :

- Thực hiện một đề án tin học là chuyển bài toán thực tế thành bài toán có thể giải quyết trên máy tính.
- Một bài toán thực tế bất kỳ đều bao gồm *dữ liệu* và *các yêu cầu xử lý trên dữ liệu* đó để xây dựng một mô hình tin học phản ánh được bài toán thực tế cần chú trọng đến hai vấn đề:
- **Tổ chức biểu diễn các đối tượng thực tế:** Mô hình tin học của bài toán, cần phải tổ chức sao cho
  - –vừa phản ánh chính xác dữ liệu thực tế,
  - –vừa dễ dàng dùng máy tính để xử lý.

  → *xây dựng cấu trúc dữ liệu.*
- **Xây dựng các thao tác xử lý dữ liệu** : Từ những yêu cầu thực tế, cần tìm ra các giải thuật tương ứng để xác định trình tự các thao tác máy tính phải thi hành để cho ra kết quả mong muốn → đây là bước *xây dựng giải thuật* cho bài toán.

2 MỐI QUAN HỆ GIỮA CTDL & GT

- **Mối quan hệ giữa cấu trúc dữ liệu và giải thuật**

Algorithms + Data Structures = Programs

Algorithms ↔ Data Structures

Giải Thuật + Cấu trúc dữ liệu = Chương trình

3 Độ phức tạp của thuật toán

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.

3 Độ phức tạp của thuật toán

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

3 Độ phức tạp của thuật toán

Sự phân lớp các thuật toán

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

Phân tích trường hợp trung bình

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

3 ĐỘ PHỨC TẠP CỦA THUẬT TOÁN

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.

3 Độ phức tạp của thuật toán

- Đánh giá độ phức tạp của thuật toán là công việc đánh giá tài nguyên các loại mà một giải thuật được sử dụng.
  - Giải thuật này thực hiện trong thời gian thế nào→Phân tích về thời gian thực hiện giải thuật
  - Giải thuật được sử dụng bao nhiêu bộ nhớ→Phân tích độ không gian nhớ mà giải thuật cần có.

3 Độ phức tạp của thuật toán

Mục tiêu của việc xác định thời gian thực hiện 1 giải thuật:

- Để ước lượng 1 chương trình sẽ thực hiện trong bao lâu
- Để ước lượng kích thước dữ liệu đầu vào lớn nhất có thể cho 1 giải thuật
- Để so sánh hiệu quả của các giải thuật khác nhau, từ đó lựa chọn 1 giải thuật thích hợp cho 1 bài toán
- Để giúp tập trung vào đoạn giải thuật được thực hiện với thời gian lớn nhất.

3 Độ phức tạp của thuật toán

- Cách thức đánh giá độ phức tạp của thuật toán:
  - Xác định độ phụ thuộc của thời gian tính của thuật toán vào kích thước của dữ liệu đầu vào.
  - Các phương pháp thực hiện:
    - Phương pháp thực nghiệm
    - Phương pháp phân tích dựa trên mô hình lý thuyết
    - Theo hướng tiệm xấp xỉ tiệm cận qua các khái niệm O().
- Để ước lượng thời gian thực hiện thuật toán xem xét 2 trường hợp
  - Trường hợp tốt nhất: T_{min}
  - Trường hợp xấu nhất: T_{max}
- Với T_{min} và T_{max} → thời gian thực hiện trung bình của thuật toán T_{avg}

3 Độ phức tạp của thuật toán

- Phương pháp thực nghiệm:
  - Cài đặt giải thuật bằng ngôn ngữ lập trình
  - Chạy chương trình với các dữ liệu vào khác nhau
  - Đo thời gian thực thi chương trình và đánh giá độ tăng trưởng so với kích thước dữ liệu đầu vào
- Hạn chế:
  - Chịu sự hạn chế của ngôn ngữ lập trình.
  - Ảnh hưởng bởi trình độ của người lập trình.
  - Chọn được các bộ dữ liệu thử đặc trưng cho tất cả tập các dữ liệu vào của thuật toán: khó khăn và tốn nhiều chi phí.
  - Sự hạn chế về số lượng và chất lượng của mẫu thử
  - Đòi hỏi môi trường kiểm thử (phần cứng và phần mềm) thống nhất, ổn định.

3 Độ phức tạp của thuật toán

- Phương pháp lý thuyết:
  - Có khả năng xem xét dữ liệu đầu vào bất kỳ
  - Sử dụng để đánh giá các giải thuật mà không phụ thuộc vào môi trường kiểm thử
  - Sử dụng với những mô tả ở mức cao của giải thuật
- Thực hiện phương pháp này cần quan tâm:
  - Ngôn ngữ mô tả thuật toán
  - Xác định độ đo thời gian tính
  - Một cách tiếp cận để khái quát hoá độ phức tạp về thời gian

3 Độ phức tạp của thuật toán

- Đánh giá giá thuật toán theo hướng tiệm xấp xỉ tiệm cận qua các khái niệm O().
- *Ưu điểm*: Ít phụ thuộc môi trường cũng như phần cứng hơn.
- *Nhược điểm*: Phức tạp.
- Các trường hợp độ phức tạp quan tâm:
  - Trường hợp tốt nhất (phân tích chính xác)
  - Trường hợp xấu nhất (phân tích chính xác)
  - Trường hợp trung bình (mang tích dự đoán)

3 Độ phức tạp của thuật toán

- Để ước lượng độ phức tạp của một thuật toán ta thường dùng khái niệm *Big-O*
- Thời gian tính của giải thuật được xác định bằng cách đếm số phép toàn cơ bản mà giải thuật thực hiện.
- Các phép toán cơ bản gồm:
  - Gán giá trị cho biến số
  - Gọi hàm hay thử tục
  - Thực hiện các phép toán số học
  - Tham chiếu vào mảng
  - Trả kết quả
  - Thực hiện các phép so sánh

$$T(n) \approx c_{op} \cdot C(n)$$

3 Độ phức tạp của thuật toán

Ví dụ:

- Bước 1. Gán Tổng = 0. Gán i = 0.
- Bước 2.
  - Tăng i thêm 1 đơn vị.
  - Gán Tổng = Tổng + i
- Bước 3. So sánh i với n
  - Nếu i < n, quay lại bước 2.
  - Ngược lại, dừng thuật toán.

```
int tong=0, i=0;
do{
        i++;
        tong+=i;
}while (i<n);
```

- Số phép gán của thuật toán là bao nhiêu?
- Số phép so sánh là bao nhiêu?

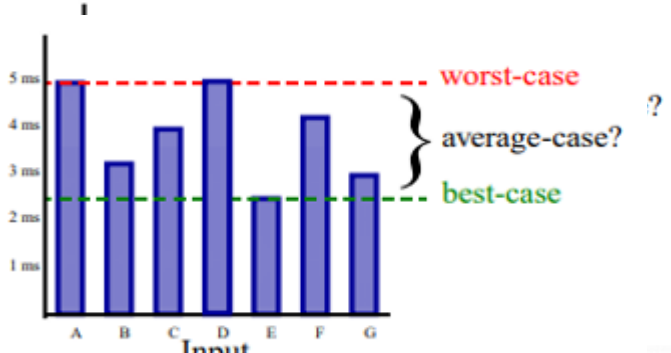
→ Gán: f(2n + 2), So sánh: f(n)

→ Độ phức tạp: O(n)

3 Độ phức tạp của thuật toán

Các độ phức tạp theo thời gian

- Thời gian tính tồi nhất (Worst-case)
  - *Thời gian nhiều nhất để thực hiện thuật toán với 1 bộ dữ liệu kích thước n*
- Thời gian tính tốt nhất (Best-case)
  - *Thời gian ít nhất để thực hiện thuật toán với 1 bộ dữ liệu cũng với kích thước n*
- Thời gian tính trung bình (Average case)
  - Thời gian trung bình cần thiết để thực hiện thuật toán trên tập hữu hạn các đầu vào kích thước n



3 Độ phức tạp của thuật toán

Các độ phức tạp thường gặp

- Ví dụ : Tìm kiếm tuần tự một giá trị trên một mảng

a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]	a[10]	a[11]	a[12]
4	8	7	10	21	14	22	36	62	91	77	81

- Thời gian xấu nhất : n
- Thời gian tốt nhất : 1
- Thời gian trung bình: $T(n) = \sum i.pi$

trong đó pi là xác suất giá trị cần tìm xuất hiện tại a[i]. pi = 1/n thì thời gian sẽ là (n+1)/2

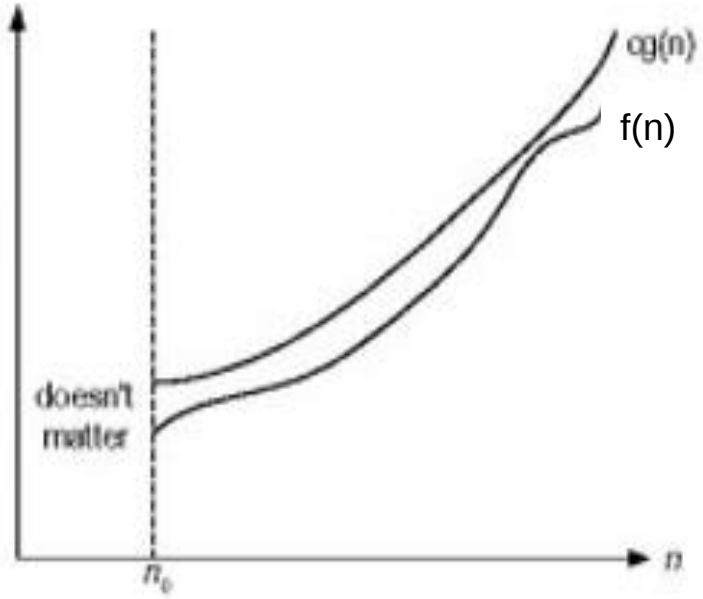
3 Độ phức tạp của thuật toán

Định nghĩa 1: Ta nói rằng f(x) = o(g(x)) khi x dần tới dương vô cùng, nếu như $\lim_{x\to+\infty} f(x)/g(x) = 0$. Khi này người ta nói rằng f(x) tăng chậm hơn so với g(x) khi x lớn dần đến +∞.

Khái niệm Big-O: Cho hàm số f(n) và g(n), ta nói rằng f(n) là O(g(n)) nếu tồn tại 2 hằng số nguyên dương c và n_0 sao cho

$f(n) <= cg(n)$ for $n >= n_0$

t(n) thuộc O(g(n))



3 Độ phức tạp của thuật toán

Định nghĩa 2: Ta nói rằng f(n) là O-lớn của g(n) khi n dần tới dương vô cùng.

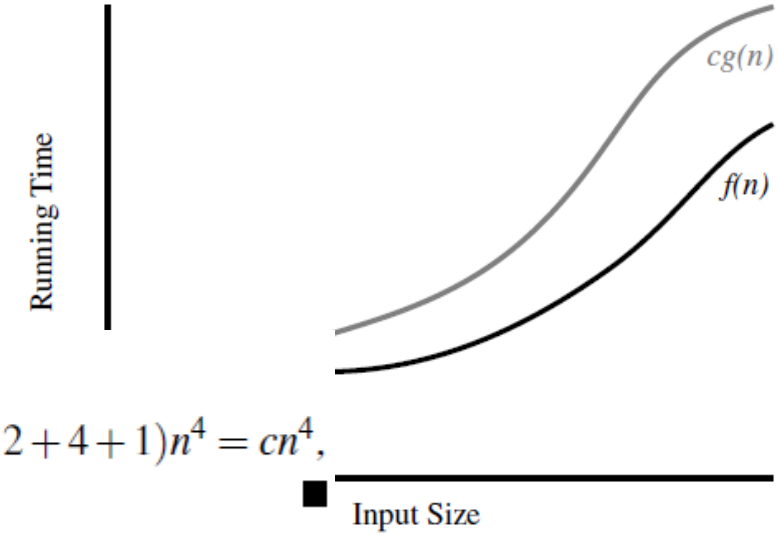
Kí hiệu f(n) = O(g(n)) hoặc đôi khi viết f(n) là O(g(n)) nếu như tồn tại hai hằng số C >0 và n_0 >0 sao cho với mọi n >= n_0 thì |f(n) | ≤C.|g(n)|.

Ví dụ 1.2. Xét hàm số f(x) = x^2+2x+3.

Rõ ràng f(x) = O(x^2),

vì với mọi x>1 ta có f(x) ≤x^2+ 2x^2.

Ngược lại ta cũng có x^2 =O(f(x)) vì x>0 ta có x^2 <f(x).



Example 3.8: $5n^4 + 3n^3 + 2n^2 + 4n + 1$ *is* $O(n^4)$.

Justification: *Note that* $5n^4 + 3n^3 + 2n^2 + 4n + 1 \leq (5+3+2+4+1)n^4 = cn^4$, *for* $c = 15$, *when* $n \geq n_0 = 1$.

■

3 Độ phức tạp của thuật toán

Định nghĩa 3: Ta nói rằng f(x) tương đương với g(x) khi x dần tới dương vô cùng, kí hiệu $f(x) \approx g(x)$, nếu như $\lim_{x \to +\infty} f(x)/g(x) = 1$.

Vídụ: $1+x+x^2 \approx x^2$, $(2x+4)^2 \approx 4x^2$

Example 3.10: $5n^2 + 3n\log n + 2n + 5$ *is* $O(n^2)$.

Justification: $5n^2 + 3n\log n + 2n + 5 \leq (5+3+2+5)n^2 = cn^2$, for $c = 15$, when $n \geq n_0 = 1$. ■

Example 3.11: $20n^3 + 10n\log n + 5$ *is* $O(n^3)$.

Justification: $20n^3 + 10n\log n + 5 \leq 35n^3$, for $n \geq 1$. ■

Example 3.12: $3\log n + 2$ *is* $O(\log n)$.

Justification: $3\log n + 2 \leq 5\log n$, for $n \geq 2$. Note that $\log n$ is zero for $n = 1$. That is why we use $n \geq n_0 = 2$ in this case. ■

Example 3.13: 2^{n+2} *is* $O(2^n)$.

Justification: $2^{n+2} = 2^n \cdot 2^2 = 4 \cdot 2^n$; hence, we can take $c = 4$ and $n_0 = 1$ in this case. ■

Example 3.14: $2n + 100\log n$ *is* $O(n)$.

Justification: $2n + 100\log n \leq 102n$, for $n \geq n_0 = 1$; hence, we can take $c = 102$ in this case. ■

3 Độ phức tạp của thuật toán

Thứ tự độ phức tạp của các thuật toán

Độ phức tạp	Thuật ngữ/tên phân lớp
O(1)	Độ phức tạp hằng số
O(logn)	Độ phức tạp logarit
O(n)	Độ phức tạp tuyến tính
O(nlogn)	Độ phức tạp nlogn
$O(n^a)$	Độ phức tạp đa thức
$O(a^n)$, a > 1	Độ phức tạp hàm mũ
O(n!)	Độ phức tạp giai thừa

3 Độ phức tạp của thuật toán

Thứ tự độ phức tạp của các thuật toán

n	$\log n$	n	$n \log n$	n^2	n^3	2^n
8	3	8	24	64	512	256
16	4	16	64	256	4,096	65,536
32	5	32	160	1,024	32,768	4,294,967,296
64	6	64	384	4,096	262,144	1.84×10^{19}
128	7	128	896	16,384	2,097,152	3.40×10^{38}
256	8	256	2,048	65,536	16,777,216	1.15×10^{77}
512	9	512	4,608	262,144	134,217,728	1.34×10^{154}

Table 3.2: Selected values of fundamental functions in algorithm analysis.

3 Độ phức tạp của thuật toán

Thứ tự độ phức tạp của các thuật toán

- Độ phức tạp hằng số, O(1). Số phép tính/thời gian chạy/dung lượng bộ nhớ không phụ thuộc vào độ lớn đầu vào.
- Độ phức tạp tuyến tính, O(n). Số phép tính/thời gian chạy/dung lượng bộ nhớ có xu hướng tỉ lệ thuận với độ lớn đầu vào. Chẳng hạn như tính tổng các phần tử của một mảng một chiều.
- Độ phức tạp đa thức, O(P(n)), với n là đa thức bậc cao (từ 2 trở lên). Chẳng hạn như các thao tác tính toán với mảng nhiều chiều (tính định thức ma trận).
- Độ phức tạp logarit, O(logn) (chú ý: bậc của nó thấp hơn so với O(n)). Chẳng hạn thuật toán Euclid để tìm ước số chung lớn nhất.
- Độ phức tạp hàm mũ, $O(2^n)$. Trường hợp này bất lợi nhất và sẽ rất phi thực tế nếu thực hiện thuật toán với độ phức tạp này.

3 Độ phức tạp của thuật toán

Cách tính độ phức tạp

Qui tắc cộng:

- Nếu T1(n) và T2(n) là thời gian thực hiện của hai đoạn chương trình P1 và P2; và T1(n)=O(f(n)), T2(n)=O(g(n) thì thời gian thực hiện của đoạn hai chương trình đó **nối tiếp nhau** là T(n)=O(**max(f(n),g(n))**)
- **Ví dụ 1-6**:
  - Lệnh gán x=15 tốn một hằng thời gian hay O(1)
  - Lệnh đọc dữ liệu scanf(“%d”, x) tốn một hằng thời gian hay O(1)
  - Vậy thời gian thực hiện cả hai lệnh trên nối tiếp nhau là O(max(1,1))=O(1)

4 Qui tắc xác định Độ phức tạp của thuật toán

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))

4 Quy tắc xác định Độ phức tạp của thuật toán

Cách tính độ phức tạp

Qui tắc nhân:

- Nếu T1(n) và T2(n) là thời gian thực hiện của hai đoạn chương trình P1 và P2 và
  T1(n) = O(f(n)), T2(n) = O(g(n) thì thời gian thực hiện của đoạn hai đoạn chương trình đó lồng nhau là T(n) = O(f(n).g(n))

4 Quy tắc xác định Độ phức tạp của thuật toán

Qui tắc tổng quát để phân tích một chương trình:

- Thời gian thực hiện của mỗi lệnh gán, Scanf, Printf là O(1)
- Thời gian thực hiện của một chuỗi tuần tự các lệnh được xác định bằng qui tắc cộng. Như vậy thời gian này là thời gian thi hành một lệnh nào đó lâu nhất trong chuỗi lệnh.
- Thời gian thực hiện cấu trúc IF là thời gian lớn nhất thực hiện lệnh sau IF hoặc sau ELSE và thời gian kiểm tra điều kiện. Thường thời gian kiểm tra điều kiện là O(1).
- Thời gian thực hiện vòng lặp là tổng (trên tất cả các lần lặp) thời gian thực hiện thân vòng lặp. Nếu thời gian thực hiện thân vòng lặp không đổi thì thời gian thực hiện vòng lặp là tích của số lần lặp với thời gian thực hiện thân vòng lặp.

4 Quy tắc xác định Độ phức tạp của thuật toán

Qui tắc tổng quát để phân tích một chương trình:

Ví dụ : Tính thời gian thực hiện của đoạn chương trình

```
void BubleSort(int a[], int N )
{           int     i, j, temp;
    {1}     for (i = 0 ; i<N-1 ; i++)
    {2}         for (j =N-1; j >i ; j --)
    {3}             if(a[j]< a[j-1]) // nếu sai vị trí thì đổi chỗ
    {4}             {       temp = a[j];
     {5}                    a[j] = a[j-1];
     {6}                    a[j-1] = temp;
                    }
}
```

Cả ba lệnh đổi chỗ {4} {5} {6} tốn O(1) thời gian, do đó lệnh {3} tốn O(1).

Vòng lặp {2} thực hiện (n-i) lần, mỗi lần O(1) do đó vòng lặp {2} tốn O((n-i).1)=O(n-i).

Vòng lặp {1} lặp (n-1) lần vậy độ phức tạp của giải thuật là $T(n) = \sum_{i=1}^{n-1}(n-i) = \frac{n(n-1)}{2} = O(n^2)$

5 Các ví dụ về Độ phức tạp của thuật toán

PHÉP TOÁN TÍCH CỰC (BEST PROXY)

- Ví dụ 1:

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

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

- 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 2:

```
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

5 Các ví dụ Độ phức tạp của thuật toán

PHÉP TOÁN TÍCH CỰC (BEST PROXY)

- Ví dụ 3:
- 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 4:

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

=> Dùng quy tắc nhân ta có $O(n^2)$

5 Các ví dụ Độ phức tạp của thuật toán

PHÉP TOÁN TÍCH CỰC (BEST PROXY)

- Ví dụ 5:

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

=> Dùng quy tắc nhân ta có O(n*m)

- Ví dụ 6:

```
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))

5 Các ví dụ Độ phức tạp của thuật toán

PHÉP TOÁN TÍCH CỰC (BEST PROXY)

- Ví dụ 7:

```
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))

5 Các ví dụ Độ phức tạp của thuật toán

PHÉP TOÁN TÍCH CỰC (BEST PROXY)

- Ví dụ 8:

```
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

5 Các ví dụ Độ phức tạp của thuật toán

- **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))

Các ví dụ trong giáo trình trang 132

7 Tổng kết

Câu hỏi và Bài tập

1. Trình bày tầm quan trọng của Cấu trúc dữ liệu và Giải thuật đối với người lập trình?
2. Các tiêu chuẩn để đánh giá cấu trúc dữ liệu và giải thuật?
3. Khi xây dựng giải thuật có cần thiết phải quan tâm tới cấu trúc dữ liệu hay không? Tại sao?
4. Liệt kê các kiểu dữ liệu cơ sở, các kiểu dữ liệu có cấu trúc trong C, Pascal?
5. Sử dụng các kiểu dữ liệu cơ bản trong C, hãy xây dựng cấu trúc dữ liệu để lưu trữ trong bộ nhớ trong (RAM) của máy tính đa thức có bậc tự nhiên n (0 =< n <= 100) trên trường số thực (ai , x $\in$ R):

$$fn(x) = \sum_{i=0}^{n} a_i x^i$$

Với cấu trúc dữ liệu được xây dựng, hãy trình bày thuật toán và cài đặt chương trình để thực hiện các công việc sau:

- Nhập, xuất các đa thức.
- Tính giá trị của đa thức tại giá trị x0 nào đó.
- Tính tổng, tích của hai đa thức.

7 Tổng kết

Câu hỏi và Bài tập

6. Cho bảng giờ tàu đi từ ga Saigon đến các ga như sau (ga cuối là ga Hà nội):

TÀU ĐI	S2	S4	S6	S8	S10	S12	S14	S16	S18	LH2	SN2
HÀNH TRÌNH	32 giờ	41 giờ	41 giờ	41 giờ	41 giờ	41 giờ	41 giờ	41 giờ	41 giờ	27giờ	10g30
SAIGON ĐI	*21g00*	*21g50*	*11g10*	*15g40*	*10g00*	*12g30*	*17g00*	*20g00*	*22g20*	*13g20*	*18g40*
MƯƠNG MÁN		2g10	15g21	19g53	14g07	16g41	21g04	1g15	3g16	17g35	22g58
THÁP CHÀM		5g01	18g06	22g47	16g43	19g19	0g08	4g05	6g03	20g19	2g15
NHA TRANG	*4g10*	*6g47*	*20g00*	*0g47*	*18g50*	*21g10*	*1g57*	*5g42*	*8g06*	*22g46*	*5g15*
TUY HÒA		9g43	23g09	3g39	21g53	0g19	5g11	8g36	10g50	2g10	
DIÊU TRÌ	*8g12*	*11g49*	*1g20*	*5g46*	*0g00*	*2g30*	*7g09*	*10g42*	*13g00*	*4g15*	
QUẢNG NGÃI		15g41	4g55	9g24	3g24	5g55	11g21	14g35	17g04	7g34	
TAM KỲ			6g11	10g39	4g38	7g10	12g40	16g08	18g21	9g03	
ĐÀ NẴNG	*13g27*	*19g04*	*8g29*	*12g20*	*6g19*	*9g26*	*14g41*	*17g43*	*20g17*	*10g53*	
HUẾ	*16g21*	*22g42*	*12g29*	*15g47*	*11g12*	*14g32*	*18g13*	*21g14*	*23g50*	*15g10*	
ĐÔNG HÀ		0g14	13g52	17g12	12g42	16g05	19g38	22g39	1g25		
ĐỒNG HỚI	19g15	2g27	15g52	19g46	14g41	17g59	21g38	0g52	3g28		
VINH	*23g21*	*7g45*	*21g00*	*1g08*	*20g12*	*23g50*	*2g59*	*7g07*	*9g20*		
THANH HÓA		10g44	0g01	4g33	23g09	3g33	6g39	9g59	12g20		
NINH BÌNH		12g04	1g28	5g54	0g31	4g50	7g57	11g12	13g51		
NAM ĐỊNH		12g37	2g01	6g26	1g24	5g22	8g29	11g44	14g25		
PHỦ LÝ		13g23	2g42	7g08	2g02	6g00	9g09	12g23	15g06		
ĐẾN HÀ NỘI	*5g00*	*14g40*	*4g00*	*8g30*	*3g15*	*7g10*	*10g25*	*13g45*	*16g20*		

7 Tổng kết

Câu hỏi và Bài tập

6. Sử dụng các kiểu dữ liệu cơ bản, hãy xây dựng cấu trúc dữ liệu thích hợp để lưu trữ

bảng giờ tàu trên vào bộ nhớ trong và bộ nhớ ngoài (disk) của máy tính.

Với cấu trúc dữ liệu đã được xây dựng ở trên, hãy trình bày thuật toán và cài đặt

chương trình để thực hiện các công việc sau:

- Xuất ra giờ đến của một tàu T_0 nào đó tại một ga G_0 nào đó.
- Xuất ra giờ đến các ga của một tàu T_0 nào đó.
- Xuất ra giờ các tàu đến một ga G_0 nào đó.
- Xuất ra bảng giờ tàu theo mẫu ở trên.

Lưu ý:

- Các ô trống ghi nhận tại các ga đó, tàu này không đi đến hoặc chỉ đi qua mà không dừng lại.
- Dòng "HÀNH TRÌNH" ghi nhận tổng số giờ tàu chạy từ ga Saigon đến ga Hà nội.

THANKS YOU !