CẤU TRÚC DỮ LIỆU, GIẢI THUẬT, PHÂN TÍCH
GIẢI THUẬT

- Khái niệm cấu trúc dữ liệu và giải thuật
- Ví dụ về giải thuật
- Ngôn ngữ diễn đạt giải thuật
- Phân tích giải thuật

Khái niệm

- Cấu trúc dữ liệu
  - Cách thức tổ chức dữ liệu để giải quyết vấn đề
- Giải thuật
  - Phác thảo, các thủ tục tính toán, các hướng dẫn từng bước để giải quyết vấn đề

"Thuật toán là một chuỗi hữu hạn các lệnh. Mỗi lệnh có một ngữ nghĩa rõ ràng và có thể được thực hiện với một lượng hữu hạn tài nguyên trong một khoảng hữu hạn thời gian"

- Chương trình
  - Thực thi thuật toán bằng 1 ngôn ngữ lập trình

Phát triển giải thuật

- Định nghĩa vấn đề rõ ràng
  - Input – Output (xem xét các trường hợp có thể)
- Đưa ra 1 phác thảo sơ bộ
  - Độc lập ngôn ngữ lập trình
- Chuyển phác thảo thành bản thực thi
  - Việc biểu diễn vấn đề (cấu trúc dữ liệu) ảnh hưởng tới vấn đề thực thi

Biểu diễn thuật toán

- Ngôn ngữ diễn đạt giải thuật:
  - Sử dụng nhiều cấp độ từ ngôn ngữ gần tự nhiên đến tựa ngôn ngữ lập trình
- Sơ đồ khối
  - Ký hiệu đồ họa

Ví dụ 1: Tìm kiếm

- INPUT
  - Một chuỗi n số nguyên (n>0)
  - q: số nguyên bất kỳ
  - $a_1, a_2, a_3, \ldots, a_n$; q

  

  - 2 5 6 10 11; 5
  - 2 5 6 10 11; 9
- OUTPUT
  - Chỉ số của q trong chuỗi a hoặc null

  j

  

  2

  null

Ví dụ 1: Tìm kiếm

search1
INPUT: A[1..n] (un)sorted array of integers, q an integer.
OUTPUT: index j such that A[j]=q. *NIL* if for all j ($1 \leq j \leq n$): A[j] $\neq q$

```
j := 1
while j ≤ n and A[j] ≠ q do j++
if j ≤ n then return j
else return NIL
```

- Chương trình được viết mã giả (pseudo-code)
- Giải thuật vét cạn: quét toàn bộ chuỗi a 1 cách tuần tự

Ví dụ 1: Tìm kiếm

- Mã nguồn C:

```c
#include <stdio.h>
#define n 5


int j, q;
int a[n] = { 11, 1, 4, -3, 22 };
int main() {
  j = 0;  q = -2;
  while (j < n && a[j] != q) { j++; }
  if (j < n) { printf("%d\n", j); }
  else { printf("NIL\n"); }
}
```

Ví dụ 1: Tìm kiếm

- Giải thuật 2: Duyệt qua chuỗi, gán giá trị cho con trỏ nếu tìm thấy

search2
INPUT: A[1..n] (un)sorted array of integers, q an integer.
OUTPUT: index j such that A[j]=q. *NIL* if for all j ($1 \leq j \leq n$): A[j] $\neq q$

```
ptr := NIL;
for j := 1 to n do
  if a[j] = q then ptr := j
return ptr;
```

Ví dụ 1: Tìm kiếm

- So sánh 2 giải thuật:
  - Search1: Quét chuỗi và so sánh. Dừng ngay khi tìm thấy.
  - Search2: Quét chuỗi từ đầu đến cuối, nếu tìm thấy ghi lại thông tin.
- Hiệu quả?
- Kết quả?

Đánh giá giải thuật

- Thuật toán đơn giản, dễ thực hiện, tốn thời gian, tài nguyên
  - Phù hợp với chương trình chỉ chạy 1 vài lần
- Thuật toán phức tạp, chạy nhanh, tốn ít tài nguyên
  - Phù hợp với chương trình chạy nhiều lần
- Thực hiện các phiên bản đơn giản trước
  - Phân tích, đánh giá, cải tiến

VD: Thuật toán kiểm tra số nguyên tố

- Prime 1:

```
for k:=2 to n-1 do
    if (n mod k=0) then return false;
return true;
```

  - Số phép tính (mod): n-2
  - Siêu máy tính thực hiện 100.000 tỷ phép tính/giây (10^{14})
  - Kiểm tra số có 25 chữ số mất $10^{25}/10^{14}*60*60*24*365$ ~ 3.170 năm!

VD: Thuật toán kiểm tra số nguyên tố

- Prime 2:

```
for k:=2 to sqrt(n) do
    if (n mod k=0) then return false;
return true;
```

  - Kiểm tra số có 25 chữ số mất sqrt(10^{25})/10^{14} ~ 0.03 giây!

Thời gian thực hiện chương trình

- Dữ liệu đầu vào
- Chất lượng mã chương trình (mã nguồn ->mã máy)
- Tốc độ thực thi lệnh của máy
- Độ phức tạp của thuật toán

Dữ liệu đầu vào

- Kích thước dữ liệu
  - Kích thước n -> hàm T(n)
  - Đơn vị T(n): Tổng số lệnh thực hiện
- Tính chất dữ liệu
  - Thời gian nhanh nhất/chậm nhất -> T(n) tốt nhất/xấu nhất
  - Thời gian trung bình: Khó tính nhất

Độ phức tạp thuật toán

- Chất lượng mã và tốc độ lệnh tùy thuộc hệ thống:
  - Không thể tính thời gian cụ thể
  - Tỷ lệ với hàm nào đó của n
- Ký hiệu O(n)
  - Cấp độ tăng của n
  - $T(n) = O(n^2)$: Tồn tại các hằng số dương c và n_0 sao cho $T(n) \leq cn^2$ với $n \geq n_0$.
  - T(n) = O(f(n)): Tồn tại các hằng số dương c và n0 sao cho T(n) < c.f(n) với n ≥ n0
- Đánh giá thời gian chạy:
  - Bỏ qua hằng số
  - Khi đó $O(n^2)$ tốt hơn $O(n^3)$. VD $100n^2 < 50n^3$ với n lớn.

Tính toán thời gian

- Quy tắc chung
  - Các lệnh đơn (gán, cộng, trừ …): O(1)
  - Lệnh tuần tự: Quy tắc cộng (= thời gian lệnh lớn nhất)
  - Lệnh If: Kiểm tra điều kiện ->O(1)
  - Vòng lặp: Quy tắc nhân số lần thực hiện bước lặp và thời gian thực hiện các lệnh ở thân vòng lặp
  - Vòng lặp lồng nhau: Quy tắc nhân số bước lặp vòng ngoài với số bước lặp vòng trong

Tính toán thời gian

- Nguyên tắc cộng cấp độ tăng của hàm:
  - $T_1(n)$ và $T_2(n)$: Thời gian chạy của 2 đoạn chương trình P_1 và P_2
  - $T_1(n) = O(f(n))$ và $T_2(n) = O(g(n))$.
  - Thời gian thực hiện của 2 đoạn chương trình nối tiếp P_1, P_2: $O(\max(f(n), g(n)))$.
  - Sử dụng khi chương trình có các đoạn tuần tự

Tính toán thời gian

- Nguyên tắc nhân cấp độ tăng của hàm:
  - $T_1(n)$ và $T_2(n)$: Thời gian chạy của 2 đoạn chương trình P_1 và P_2
  - $T_1(n) = O(f(n))$ và $T_2(n) = O(g(n))$.
  - Thời gian thực hiện của 2 đoạn chương trình **lồng nhau** P_1, P_2: $O(f(n)*g(n))$.
  - Sử dụng khi chương trình có các đoạn tuần tự

Tính toán thời gian

- Thuật toán sắp xếp nổi bọt (bubble sort)

```
void bubble (int a[n]){
   int i, j, temp;
1. for (i = 0; i < n-1; i++)
2.       for (j = n-1; j>= i+1; j--)
3.               if (a[j-1] > a[j]{
                         // Đổi chỗ cho a[j] và a[j-1]
4.                       temp = a[j-1];
5.                       a[j-1] = a[j];
6.                       a[j] = temp;
                 }
}
```

Tính toán thời gian

- Thuật toán sắp xếp nổi bọt (bubble sort)
  - Kích thước dữ liệu: n (số phần tử cần sắp)
  - Lệnh gán: Thời gian thực hiện cố định, ko phụ thuộc n -> O(1)
  - Lệnh If: Kiểm tra điều kiện ->O(1)
  - Vòng lặp trong: Số bước lặp n-i -> O(n-i), mỗi bước lặp thực hiện lệnh kiểm tra -> O(1). Tổng: O((n-i)*1)=O(n-i)
  - Vòng lặp ngoài: (n-1) lần -> O(n-1)
  - Tổng thời gian: $\sum (n-i) = n(n-1)/2 = n^2/2 - n/2 = O(n^2)$