CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

MỘT SỐ VẤN ĐỀ CƠ BẢN CỦA CẤU TRÚC DỮ LIỆU VÀO GIẢI THUẬT

Mục lục

Khái niệm về kiểu và cấu trúc dữ liệu

- Cấu trúc dữ liệu: Cách tổ chức dữ liệu trên máy tính để thuận tiện cho các tính toán
- Thuật toán: Một thủ tục xác định bao gồm 1 dãy các thao tác tính toán để thu được kết quả đầu ra với mỗi dữ liệu đầu vào xác định

Khái niệm về kiểu và cấu trúc dữ liệu

- Kiểu dữ liệu
  - Tập các giá trị
  - Tập các phép toán thao tác trên các giá trị
  - Biểu diễn và cài đặt trên máy tính
- Ví dụ kiểu số nguyên int
  - Tập giá trị -2^{31} đến $2^{31} - 1$
  - Tập phép toán: +, -, *, / mod
- Kiểu dữ liệu trừu tượng (Abstract Data Type - ADT)
  - Tập các giá trị
  - Tập tác thao tác
  - Chưa quan tâm đến biểu diễn và cài đặt trên máy tính

Thuật toán và một số vấn đề liên quan

- Các bài toán tối ưu tổ hợp
  - Lập lịch, lập lộ trình vận tải, thời gian biểu, lập kế hoạch sản xuất,. . .
- Xử lý ảnh, thị giác máy tính, học máy, phân tích dữ liệu
- Cơ sở dữ liệu
- Các bài toán quản lý bộ nhớ, lập lịch thực hiện tiến trình trong các hệ điều hành
- …

Phương pháp biểu diễn thuật toán

- Giả mã: Ngôn ngữ để biểu diễn thuật toán một cách thân thiện, ngắn gọn mà không cần viết chương trình (bằng 1 ngôn ngữ lập trình cụ thể)

```
max(a[1..n]) {
  m = a[1];
  for i = 2 to n do {
    if(m < a[i])
      m = a[i];
  }
  return m;
}
```

Độ phức tạp của thuật toán

- Phân tích độ phức tạp của thuật toán
  - Thời gian
  - Bộ nhớ
- Thực nghiệm
  - Viết chương trình hoàn chỉnh bằng một ngôn ngữ lập trình
  - Chạy chương trình với các bộ dữ liệu khác nhau
  - Đo thời gian thực hiện chương trình và vẽ biểu đồ
- Nhược điểm của phương pháp thực nghiệm
  - Cần lập trình
  - Kết quả thực nghiệm không bao quát được hết các trường hợp
  - Cùng 1 chương trình nhưng chạy trên 1 cấu hình máy khác nhau sẽ cho kết quả khác nhau

Độ phức tạp của thuật toán

- Phân tích độ phức tạp thời gian tính như là một hàm của kích thước dữ liệu đầu vào
- Kích thước dữ liệu đầu vào
  - Số bít cần để biểu diễn dữ liệu đầu vào
  - Mức cao: số phần tử của dãy, ma trận đầu vào
- Câu lệnh cơ bản
  - Thực hiện trong thời gian hằng số và không phụ thuộc kích thước dữ liệu đầu vào
  - Ví dụ: các phép toán cơ bản: +, -, *, /, so sánh,…

Độ phức tạp của thuật toán

- Phân tích độ phức tạp thời gian
  - Đếm số câu lệnh cơ bản được thực hiện như một hàm của kích thước dữ liệu đầu vào

```
sum(a[1..n]) {
  s = a[1];
  for i = 2 to n do {
    s = s + a[i];
  }
  return s;
}
```

Số câu lệnh cơ bản của hàm **`sum(a[1..n])`** là cỡ **`n`**

Độ phức tạp của thuật toán

- Ký hiệu tiệm cận (O lớn)
  - Dùng để viết ngắn gọn hàm độ phức tạp thời gian
  - Thể hiện độ tăng của hàm độ phức tạp thời gian theo kích thước dữ liệu đầu vào
  - $f(n) = O(g(n))$: độ tăng của $f(n)$ không vượt quá độ tăng của $g(n)$, nói cách khác $\lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty$
  - $f(n) = \Theta(g(n))$: độ tăng của $f(n)$ bằng độ tăng của $g(n)$, nói cách khác $0 < \lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty$
  - Ví dụ
    - $2n^2 + 10^6 n + 5 = O(n^2)$
    - $10^3 n\log n + 2n + 10^4 = O(n\log n)$
    - $10^3 n\log n + 2n + 10^4 = O(n^3)$
    - $2^n + n^{10} + 1 = O(2^n)$
    - Độ phức tạp về thời của hàm **`sum(a[1..n])`** là O(**n**)

Độ phức tạp của thuật toán

- Câu lệnh cơ bản trong hàm **`sort`** là khối các câu lệnh so sanh và đổi chỗ 2 phần tử **`a[i]`** và **`a[j]`**
- Số câu lệnh cơ bản là **`n(n-1)/2`** = O(n^2)
- Độ phức tạp về thời của hàm **`sort(a[1..n])`** là O(n^2)

```
sort(a[1..n]) {
  for i = 1 to n-1 do
    for j = i+1 to n do
      if(a[i] > a[j]) {
        tmp = a[i];
        a[i] = a[j];
        a[j] = tmp;
      }
}
```

Độ phức tạp của thuật toán

- Trong nhiều trường hợp, với cùng kích thước dữ liệu đầu vào, các bộ dữ liệu khác nhau sẽ cho độ phức tạp về thời gian tính khác nhau
  - Thời gian tính trong tình huống tồi nhất (worst-case time complexity): Bộ dữ liệu cho thời gian tính lâu nhất
  - Thời gian tính trong tình huống tốt nhất (best-case time complexity): Bộ dữ liệu cho thời gian tính nhanh nhất
  - Thời gian tính trung bình: trung bình về thời gian tính của tất cả các bộ dữ liệu đầu vào với cùng kích thước xác định

Ví dụ mở đầu

- Bài toán dãy con cực đại
  - Đầu vào: Cho dãy $a = a_1, a_2, \ldots, a_n$. Một dãy con của a là dãy gồm một số liên tiếp các phần tử $a_i, a_{i+1}, \ldots, a_j$ và có trọng số là $a_i + a_{i+1} + \ldots + a_j$
  - Đầu ra: Tìm dãy con có trọng số lớn nhất của dãy a
  - Ví dụ: dãy a = 2, 4, -7, 5, 7, -10, 4, 3, dãy con cực đại của a là dãy 5, 7

Ví dụ mở đầu

- Thuật toán 1 (`subseq1`)
  - Duyệt tất cả các dãy con
  - Tính tổng các phần tử của mỗi dãy con và giữ lại dãy con có trọng số lớn nhất
  - Độ phức tạp O(n^3)

```
subseq1(a[1..n]){
  max = -∞;
  for i = 1 to n do{
    for j = i to n do{
      s = 0;
      for k = i to j do
        s = s + a[k];
      max = s > max ? s : max;
    }
  }
  return max;
}
```

Ví dụ mở đầu

- Thuật toán 2 (**`subseq2`**)
  - Duyệt tất cả các dãy con
  - Tính tổng các phần tử của mỗi dãy con và giữ lại dãy con có trọng số lớn nhất
  - Tận dụng tính chất liên tiếp để tính tổng dãy con dựa vào dãy con trước đó
  - Độ phức tạp $O(n^2)$

```
subseq1(a[1..n]){
  max = -∞;
  for i = 1 to n do{
    s = 0;
    for j = i to n do{
      s = s + a[k];
      max = s > max ? s : max;
    }
  }
  return max;
}
```

Ví dụ mở đầu

- Thuật toán 3 (`subseq3`)
  - Chia dãy đã cho thành 2 dãy con độ dài đều nhau
  - Tìm dãy con lớn nhất của dãy bên phải
  - Tìm dãy con lớn nhất của dãy bên trái
  - Tìm dãy con lớn nhất có 1 phần thuộc dãy con bên phải và 1 phần thuộc dãy con bên trái
  - Độ phức tạp O(*n*log*n*)

```
subseq3(a[1..n], l, r){
  if(l = r) return a[r];
  i = (l+r)/2;
  ml = subseq3(a,l,i);
  mr = subseq3(a,i+1,r);
  mlr = maxLeft(a,l,i) +
        maxRight(a,i+1,r);
  max = mlr;
  max = max < ml ? ml : max;
  max = max < mr ? mr : max;
  return max;
}
```

Ví dụ mở đầu

```
maxLeft(a[1..n], l, r){
  max = -∞;
  s = 0;
  for i = r downto l do{
    s = s + a[i];
    if(s > max) max = s;
  }
  return max;
}
```

```
maxRight(a[1..n], l, r){
  max = -∞;
  s = 0;
  for i = l to r do{
    s = s + a[i];
    if(s > max) max = s;
  }
  return max;
}
```

Ví dụ mở đầu

- Thuật toán 4 (**subseq4**)
  - Dựa trên quy hoạch động
  - S_i là trọng số dãy con lớn nhất của dãy $a_1, \ldots, a_i$ mà phần tử cuối cùng là a_i ($\forall i = 1,\ldots, n$)
  - $S_1 = a_1$
  - $S_i = \begin{cases} S_{i-1} + a_i, \text{ nếu } S_{i-1} > 0 \\ a_i, \text{ nếu } S_{i-1} \leq 0 \end{cases}$
  - Độ phức tạp O(n)

```
subseq4(a[1..n]){
  max = -∞;
  s = a[1];
  for i = 2 to n do{
    if(s > 0)
      s = s + a[i];
    else s = a[i];
    if(s > max) max = s;
  }
  return max;
}
```