KHOA CÔNG NGHỆ THÔNG TIN
ĐH. KHOA HỌC TỰ NHIÊN TP.HCM

BỘ MÔN KHOA HỌC MÁY TÍNH

Đồ án Nhập Môn Phân Tích Độ Phức Tạp Thuật Toán

Đề tài: Đánh giá các thuật toán Sort

Giảng viên hướng dẫn lý thuyết

TS. Trần Đan Thư

Giảng viên hướng dẫn thực hành

Dương Chí Nhân

Nhóm thực hiện

0712002 – Nguyễn Thanh Đồng

0712228 – Trần Trung Kiên

0712394 – Bành Trí Thành

0712474 – Nguyễn Trọng Nhật Trung

0712476 – Phan Thanh Trí

SELECTION SORT

Ý tưởng thuật toán

- Ta chọn phần tử nhỏ nhất trong N phần tử ban đầu, đưa phần tử này về đầu dãy hiện hành. Sau đó, ta không quan tâm đến nó nữa, ta xem dãy hiện hành chỉ còn N-1 phần tử của dãy ban đầu tính từ vị trí thứ 2. Cứ vậy, cho đến khi dãy hiện hành chỉ còn 1 phần tử, ta được 1 dãy sắp tăng.
- Các bước tiến hành như sau:
  - Bước 1: Khởi động i = 1
  - Bước 2: Tìm phần tử nhỏ nhất a[min] trong dãy hiện hành từ a[i] đến a[N]
  - Bước 3: Hoán vị a[min] và a[i]
  - Bước 4: i = i+1
    - Nếu i < =N-1: quay trở lại bước 2
    - Ngược lại: STOP!

Độ phức tạp

Để chọn được phần tử nhỏ nhất, ta cần duyệt qua n phần tử (tốn n-1 phép so sánh) và sau đó hoán vị nó với phần tử đầu tiên của dãy hiện hành. Để tìm phần tử nhỏ nhất tiếp theo, ta cần duyệt qua n-1 phần tử (tốn n-2 phép so sánh). Cứ như vậy, ta thấy ngay thuật toán sẽ tốn (n-1) + (n-2) + … + 1 = n(n-1)/2 = $O(n^2)$ phép so sánh. Mỗi lần duyệt, ta luôn phải hoán vị 1 lần (1 hoán vị tương đương với 3 phép gán), nghĩa là thuật toán sẽ tốn 3(n-1) + 3(n-2) + … + 3 = 3n(n-1)/2 = $O(n^2)$ phép gán.

Tổng kết lại, ta luôn có độ phức tạp của thuật toán Selection Sort thuộc $O(n^2)$ trong mọi trường hợp.

INTERCHANGE SORT

Ý tưởng thuật toán

- Ý tưởng chính của thuật toán này là ta tìm các cặp nghịch thế và triệt tiêu chúng. Ta xuất phát từ phần tử đầu tiên của dãy, tìm tất các các cặp nghịch thế chứa phần tử này, triệt tiêu chúng bằng các hoán vị phần tử này với phần tử tương ứng trong cặp nghịch thế. Ta dễ nhận thấy sau lần duyệt đầu tiên, phần tử đầu tiên chính là phần tử nhỏ nhất của dãy. Ta tiếp tục xử lý với phần tử thứ hai, ta có được phần tử thứ hai chính là phần tử nhỏ thứ hai của dãy. Cứ như vậy, sau khi xử lý với phần tử thứ N -1 của dãy, ta được một dãy sắp tăng.
- Các bước tiến hành như sau:
  - Bước 1: Khởi động i = 1
  - Bước 2: j = i+1
  - Bước 3: Trong khi j <= N thực hiện:
    - Nếu a[i]>a[j]: Hoán vị a[i] và a[j]
    - j = j+1
  - Bước 4: i = i+1
    - Nếu i <= N-1: quay trở lại bước 2
    - Ngược lại: STOP!

Độ phức tạp

- Thấy ngay số phép so sánh là luôn không đổi, tức không phụ thuộc vào tình trạng ban đầu của dãy. Ta có thể ước lượng số phép so sánh bằng (n-1) + (n-2) + … + 1 = n(n-1)/2 (phần tử thứ i được so sánh với n-i phần tử còn lại.)

- Số phép hoán vị (tương đương 3 phép gán) lại phụ thuộc vào tình trạng ban đầu của dãy. Cụ thể như sau:
    - Trường hợp tốt nhất: Dãy ban đầu đã có thứ tự. Ta thấy ngay ta không tốn một phép hoán vị nào.
    - Trường hợp xấu nhất: Dãy ban đầu có thứ tự ngược. Ta thấy ngay mỗi lần so sánh phần tử thứ i với n-i phần tử còn lại, ta đều phải thực hiện hoán vị. Điều này có nghĩa là số phép hoán vị bằng n(n-1)/2.

Tổng kết lại, ta có độ phức tạp của Interchange Sort thuộc $O(n^2)$ trong mọi trường hợp.

BUBBLE SORT

Ý tưởng thuật toán

- Xét từ đáy và phần tử nhẹ nổi lên trên.
- Các bước thực hiện:
    - Bước 1: Khởi động i = 1
    - Bước 2: j = N //Duyệt từ cuối dãy về vị trí i

        Trong khi j>i thực hiện:
        - Nếu a[j]<a[j-1]: hoán vị a[j], a[j-1]
        - j = j - 1
    - Bước 3: i = i + 1
        - Nếu i <= N-1: quay trở lại bước 2.
        - Ngược lại: STOP!

Độ phức tạp

- Thấy ngay số phép so sánh là luôn không đổi, tức không phụ thuộc vào tình trạng ban đầu của dãy. Với i bất kỳ, ta luôn phải so sánh V[j] với V[j-1], mà j chạy từ n đến i+1, tức ta tốn n-i phép so sánh. Thêm nữa, i chạy từ 1 đến n-1.

Vậy ta tính được số phép so sánh tổng cộng: ∑(n-i) với i chạy từ 1 đến n-1 = (n-1) + (n-2) + … + 1 = n(n-1)/2.

- Số phép hoán vị (tương đương 3 phép gán) lại phụ thuộc vào tình trạng ban đầu của dãy. Cụ thể như sau:
  - Trường hợp tốt nhất: Dãy ban đầu đã có thứ tự. Ta thấy ngay ta không tốn một phép hoán vị nào.
  - Trường hợp xấu nhất: Dãy ban đầu có thứ tự ngược. Xét i bất kỳ, ta thấy rằng mỗi lần so sánh a[j] với a[j-1], ta đều phải thực hiện hoán vị. Điều này có nghĩa là số phép hoán vị bằng n(n-1)/2.

Tổng kết lại, ta có độ phức tạp của Bubble Sort thuộc $O(n^2)$ trong mọi trường hợp.

INSERTION SORT

Ý tưởng thuật toán

- Giả sử ta có dãy $a_1, a_2, …, an$ trong đó i phần tử đầu tiên $a_1, a_2, …, a_i$ đã có thứ tự. Ý tưởng của thuật toán là tìm vị trị thích hợp và chèn phần tử a_{i+1} vào dãy đã có thứ tự trên để có được một dãy mới có thứ tự. Cứ thế, làm đến cuối dãy ta sẽ được một dãy có thứ tự.
- Với dãy ban đầu $a_1, a_2, …, a_n$ ta có thể coi đoạn chỉ có một phần tử a_1 là một đoạn đã có thứ tự, sau đó ta chèn phần tử a_2 vào dãy a_1 để có dãy a_1a_2 có thứ tự. Tiếp đó, ta lại chèn phần tử a_3 vào dãy a_1a_2 để có dãy $a_1a_2a_3$ có thứ tự. Cứ thế, đến cuối cùng ta chèn phần tử a_n vào dãy $a_1a_2…a_{n-1}$ ta sẽ được dãy $a_1a_2…a_n$ có thứ tự.
- Các bước thực hiện:
  - Bước 1: Khởi động với i = 2 //Đoạn a[1] đã được sắp
  - Bước 2: x = a[i]. Tìm vị trí thích hợp pos trong đoạn a[1]…a[i-1] để chèn x vào
  - Bước 3: Dời đoạn a[pos]…a[i-1] sang phải để có chỗ đưa x vào.
  - Bước 4: a[pos] = x
  - Bước 5: i = i + 1

- Nếu i<=n: quay lại bước 2.
- Ngược lại: STOP!

Độ phức tạp

- Ta thấy các phép so sánh xảy ra trong vòng lặp nhằm tìm vị trí thích hợp pos để chèn x. Mỗi lần so sánh mà thấy vị trí đang xét không thích hợp, ta dời phần tử a[pos] sang phải.
- Ta cũng thấy số phép gán và số phép so sánh của thuật toán phụ thuộc vào tình trạng của dãy ban đầu. Do đó ta chỉ có thể ước lượng như sau:
  - Trường hợp tốt nhất: dãy ban đầu đã có thứ tự. Ta tìm được ngay vị trí thích hợp để chèn ngay lần so sánh đầu tiên mà không cần phải vô vòng lặp. Như vậy, với i chạy từ 2 đến n thì số phép so sánh tổng cộng sẽ là n-1. Còn với số phép gán, do thuật toán không chạy vào vòng lặp nên xét i bất kỳ, ta luôn chỉ phải tốn 2 phép gán(x = a[i] và a[pos] = x). Từ đây, ta tính được số phép gán tổng cộng bằng 2(n - 1).
  - Trường hợp xấu nhất: dãy ban đầu có thứ tự ngược. Ta thấy ngay vị trí thích hợp pos luôn là vị trí đầu tiên của dãy đã có thứ tự, và do đó, để tìm ra vị trí này ta phải duyệt hết dãy đã có thứ tự. Xét i bất kỳ, ta có số phép so sánh là i-1, số phép gán là (i - 1) + 2 = i + 1. Với i chạy từ 2 đến n, ta tính được số phép so sánh tổng cộng bằng 1 + 2 + … + (n - 1) = n(n - 1)/2 và số phép gán bằng 3 + 4 + .. + (n + 1) = (n + 4)(n - 1)/2

Tổng kết lại, ta có độ phức tạp của Insertion Sort như sau:

- *Trường hợp tốt nhất: O(n)*
- *Trường hợp xấu nhất $O(n^2)$*

HEAP SORT

Ý tưởng thuật toán

- Định nghĩa Heap: Heap là mảng 1 chiều chứa các phần tử từ a_1, a_2 … a_n. Các phần tử từ $a_{[n/2+1]}$ đến $a_{[n]}$ là heap tự nhiên. Các phần tử còn lại thỏa $a_{[i]} >= a_{[2*i]}$ và $a_{[i]} >= a_{[2*i+1]}$ (i=1…n/2) (Điều kiện này là cho sắp xếp tăng dần)
- Như vậy ta thấy gốc của heap luôn là phần tử lớn nhất, nếu ta lần lượt rút trích phần tử gốc và xây dựng lại heap đã bị rút trích ta sẽ có một mảng có thứ tự tăng dần.
- Giải thuật:
    - Bước 1: Xây dựng heap từ mảng ban đầu.
    - Bước 2: Hoán vị phần tử đầu và cuối mảng, rồi giảm dần số phần tử của mảng xuống thành n-1 (bỏ phần tử cuối).
    - Bước 3: Tiếp tục xây dựng và hoán vị như trên cho đến hết ta sẽ có dãy được sắp thứ tự tăng dần (muốn sắp giảm dần ta chi cần cho phần tử gốc của heap là phần tử nhỏ nhất).

Độ phức tạp

MERGE SORT

Ý tưởng thuật toán

- Cho dãy ban đầu a_1, a_2, …, a_n. Ta luôn có thể coi nó là tập hợp liên tiếp của các dãy có thứ tự. Ta gọi các dãy có thứ tự này là các dãy con.
- Trong phương pháp Merge Sort, vấn đề là ta tìm cách phân hoạch dãy ban đầu thành các dãy con. Sau khi phân hoạch xong, dãy ban đầu sẽ được tách thành hai dãy phụ theo nguyên tắc phân phối luân phiên dãy con. Sau đó, ta trộn từng cặp dãy con của hai dãy phụ thành một dãy con của dãy ban đầu. Ta nhận thấy số dãy con của dãy ban đầu lúc này giảm đi ít nhất là một nửa. Cứ thế sau một số bước, ta sẽ nhận được dãy ban đầu với số dãy con bằng 1, có nghĩa là ta đã sắp xếp xong.
- *Trộn trực tiếp:* đây là phương pháp trộn đơn giản nhất. Việc phân hoạch dãy ban đầu đơn giản như sau: Với dãy ban đầu có n phân tử, ta cứ phân hoạch

thành n dãy con. Vì rằng mỗi dãy con chỉ có 1 phần tử nên nó là dãy có thứ tự. Cứ mỗi lần tách – trộn, chiều dài của dãy con sẽ được nhân đôi.

- Các bước tiến hành:
  - Bước 1: Khởi động k = 1 //Với k là chiều dài dãy con
  - Bước 2: Phân phối dãy a = $a_1, a_2, \ldots, a_n$ vào hai dãy phụ b, c theo nguyên tắc phân phối luân phiên từng dãy con.

    b = $a_1, \ldots, a_k, a_{2k+1}, \ldots, a_{3k}, \ldots$

    c = $a_{k+1}, \ldots, a_{2k}, a_{3k+1}, \ldots, a_{4k}$
  - Bước 3: Từ 2 dãy phụ b, c, ta trộn từng cặp dãy con và đưa vào dãy a.
  - Bước 4: k = k*2
    - Nếu k<n: quay lại bước 2.
    - Ngược lại: STOP!

Độ phức tạp

- Ta thấy ngay số lần lặp của bước 2(phân phối) và bước 3(trộn) bằng $\log_2 n$. Ta cũng thấy rằng chi phí thực hiện bước 2 và bước 3 tỉ lệ thuận với n. Như vậy, ta có thể ước tính chi phí thực hiện của giải thuật Merge Sort thuộc $O(n\log_2 n)$.
- Ta nhận thấy rằng giải thuật làm việc một cách cứng nhắc, không tận dụng được tính thứ tự một phần của dãy ban đầu. Do đó, trong mọi trường hợp độ phức tạp là như nhau. Đây là một nhược điểm của phương pháp trộn trực tiếp.

BINARY TREE

Ý tưởng thuật toán

Độ phức tạp

QUICK SORT

Ý tưởng thuật toán

- QuickSort chia mảng thành hai danh sách bằng cách so sánh từng phần tử của danh sách với một phần tử được chọn được gọi là phần tử chốt. Những phần tử

nhỏ hơn hoặc bằng phần tử chốt được đưa về phía trước và nằm trong danh sách con thứ nhất, các phần tử lớn hơn chốt được đưa về phía sau và thuộc danh sách con thứ hai. Cứ tiếp tục chia như vậy tới khi các danh sách con đều có độ dài bằng 1.

- Sau 1 lượt phân hoạch ta có:
  - $V_0 \dots V_j < x$, phân hoạch tiếp $V_0 \dots V_j$.
  - $V_{j+1}\ V_{i-1} = x$
  - $V_i \dots V_{n-1} > x$, phân hoạch tiếp $V_i \dots V_{n-1}$

Độ phức tạp

Ta nhận thấy hiệu quả của thuật toán phụ thuộc vào việc chọn giá trị mốc (hay phần tử chốt).

- Trường hợp tốt nhất: mỗi lần phân hoạch ta đều chọn được phần tử median (phần tử lớn hơn hay bằng nửa số phần tử và nhỏ hơn hay bằng nửa số phần tử còn lại) làm mốc. Khi đó dãy được phân hoạch thành hai phần bằng nhau, và ta cần $\log_2(n)$ lần phân hoạch thì sắp xếp xong. Ta cũng dễ nhận thấy trong mỗi lần phân hoạch ta cần duyệt qua n phần tử. Vậy độ phức tạp trong trường hợp tốt nhất thuộc $O(n\log_2(n))$.
- Trường hợp xấu nhất: mỗi lần phần hoạch ta chọn phải phần tử có giá trị cực đại hoặc cực tiểu làm mốc. Khi đó dãy bị phân hoạch thành hai phần không đều: một phần chỉ có một phần tử, phần còn lại có n-1 phần tử. Do đó, ta cần tới n lần phân hoạch mới sắp xếp xong. Vậy độ phức tạp trong trường hợp xấu nhất thuộc $O(n^2)$.

Tổng kết lại, ta có độ phức tạp của Quick Sort như sau:

- *Trường hợp tốt nhất: $O(n\log_2(n))$*
- *Trường hợp xấu nhất: $O(n^2)$*
- *Trường hợp trung bình: $O(n\log_2(n))$*

SHELL SORT

Ý tưởng thuật toán

Độ phức tạp