Các chủ đề nâng cao về sắp xếp

anhtt-fit@mail.hut.edu.vn

Ứng dụng của sắp xếp

- Tìm kiếm thông tin: dữ liệu đã sắp xếp sẽ cho phép tìm kiếm thông tin hiệu quả và nhanh chóng hơn
- Nghiên cứu về hoạt động
  - Thời gian xử lý ngắn nhất: Giả sử có N công việc, công việc j đòi hỏi tj (s) xử lý thời gian. Chúng ta cần hoàn thành tất cả các công việc trong thời gian nhanh nhất => sắp xếp các công việc theo thứ tự hợp lý
  - Cân bằng tải: có M bộ vi xử lý và N công việc cần hoàn thành, mục tiêu là thời gian hoàn thành ngắn nhất.
- - Nén dữ liệu: mã hóa Huffman. Hiệu quả phụ thuộc vào việc sắp xếp thứ tự xuất hiện các mục
- Giải thuật xử lý xâu: tìm tiền tố dài nhất giữa một tập các xâu, tìm chuỗi chung giữa các xâu: cần giải thuật sắp xếp xâu

Các giải thuật sắp xếp

Rất nhiều

Internal sorts – sắp xếp trong

- Sắp xếp chèn, sắp xếp lựa chọn, sắp xếp nổi bọt..
- Sắp xếp nhanh, sắp xếp trộn, sắp xếp vun đống…

External sorts – sắp xếp ngoài

- Poly-phase mergesort, cascade-merge, oscillating sort. (trộn nhiều đoạn, thác nước, dao động)

Radix sorts – sắp xếp cơ số

- Distribution, MSD, LSD.
- 3-way radix quicksort.

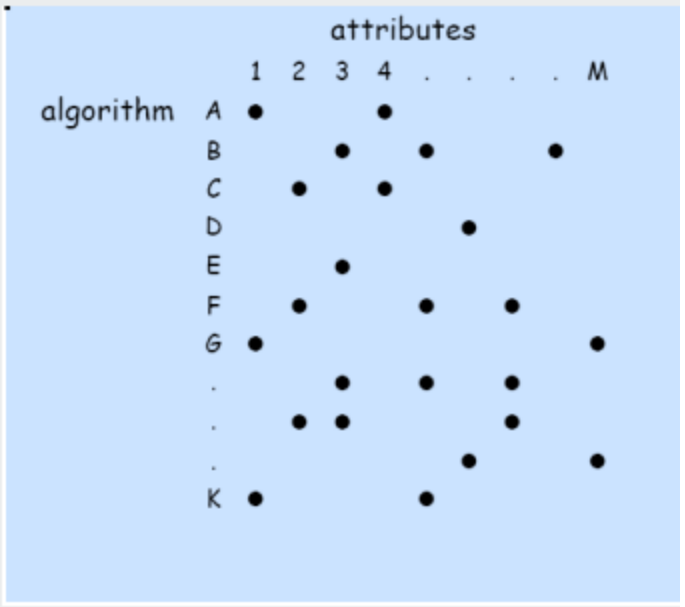
Parallel sorts – sắp xếp song song

- Bitonic sort, Batcher even-odd sort.
- Smooth sort, cube sort, column sort.
- GPUsort.

Tiêu chí lựa chọn giải thuật

Nhiều yếu tố ảnh hưởng:

- Ổn định
- Nhiều khóa
- Các khóa là phân biệt
- Nhiều dạng khóa
- Danh sách liên kết hay mảng
- Kích thước bản ghi lớn hay nhỏ
- Dữ liệu được sắp xếp ngẫu nhiên?



Không thể bao phủ tất cả các yếu tố

Tiêu chí lựa chọn giải thuật

	inplace?	stable?	worst	average	best	remarks
selection	x		$N^2 / 2$	$N^2 / 2$	$N^2 / 2$	N exchanges
insertion	x	x	$N^2 / 2$	$N^2 / 4$	N	use for small N or partially ordered
shell	x		?	?	N	tight code, subquadratic
quick	x		$N^2 / 2$	$2 N \ln N$	$N \lg N$	$N \lg N$ probabilistic guarantee fastest in practice
3-way quick	x		$N^2 / 2$	$2 N \ln N$	$N \lg N$	improves quicksort in presence of duplicate keys
merge		x	$N \lg N$	$N \lg N$	$N \lg N$	$N \lg N$ guarantee, stable
???	x	x	$N \lg N$	$N \lg N$	$N \lg N$	holy sorting grail

Ví dụ 1

Vấn đề

- Sắp xếp một file lớn có thứ tự ngẫu nhiên các bản ghi kích thước nhỏ

Ví dụ

- Xử lý các bản ghi giao dịch của công ty điện thoại

Giải thuật sắp xếp nào sử dụng:

1. Quicksort: YES, nó được thiết kế cho vấn đề này
2. Insertion sort: No, thời gian bậc 2 cho thứ tự ngẫu nhiên
3. Selection sort: No, thời gian luôn là bậc 2

Ví dụ 2

Vấn đề

- Sắp xếp một file kích thước lớn, gần như đã theo trật tự

Ví dụ

- Sắp xếp lại CSDL lớn sau vài lần thay đổi

Giải thuật sắp xếp nào sử dụng:

1. Quicksort: No
2. Insertion sort: YES, thời gian là tuyến tính
3. Selection sort: No, thời gian luôn là bậc 2

Ví dụ 3

Vấn đề: sắp xếp một file các bản ghi lớn, có các khóa nhỏ

Ví dụ: tổ chức lại các file MP3, bản ghi = kích thước file MP3, khóa có thể là tên file…

Giải thuật sắp xếp nào sử dụng:

1. Mergesort: No
2. Insertion sort: No
3. Selection sort: YES, thời gian tuyến tính

Ex: 5,000 bản ghi, mỗi bản ghi 2 triệu bytes, với 100 byte khóa.

- Chi phí so sánh: $100 \times 5000^2 / 2 = 1.25$ billion
- Chi phí đổi chỗ: 2,000,000 x 5,000 = 10 trillion
- Mergesort có thể chậm hơn log (5000) lần

Trùng lặp khóa

Mục đích: Nhóm các bản ghi có cùng khóa

- Sắp xếp dân số theo tuổi
- Tìm các điểm thẳng hàng
- Loại bỏ các thư trùng lặp
- Sắp xếp hồ sơ lao động theo trường theo học

Các đặc trưng điển hình của những ứng dụng này:

- File lớn
- Số lượng nhỏ các giá trị

3-Way Partitioning – Phân chia 3 phần

Phân chia các phần tử làm 3 phần

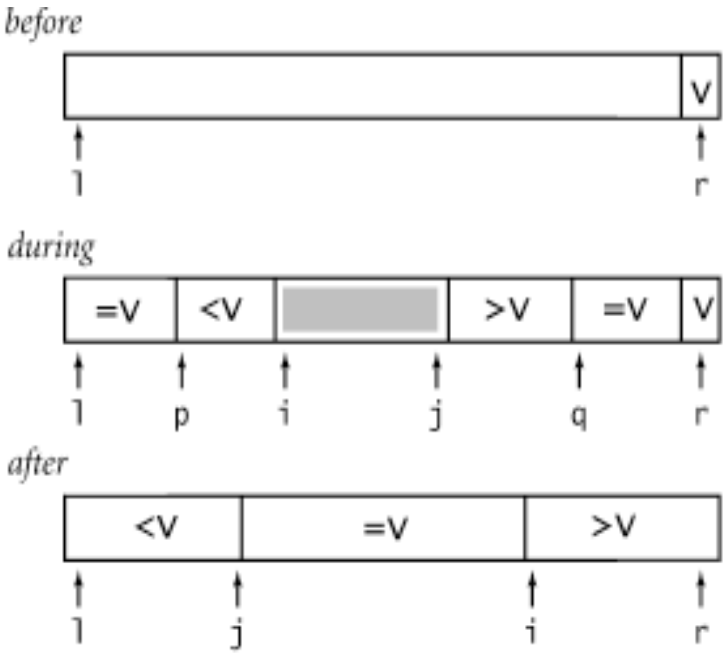
- Các phần tử giữa i và j, bằng với phần tử phân chia v
- Không có phần tử nào lớn hơn v ở bên trái i
- Không có phần tử nào nhỏ hơn v ở bên phải j

Thực hiện

3-way partitioning (Bentley-McIlroy): Phân chia thành 4 phần:

- Không có phần tử nào lớn hơn v ở bên trái của i
- Không có phần tử nào nhỏ hơn v ở bên phải của j
- Các phần tử bằng v ở bên trái của p
- Các phần tử bằng v ở bên phải của q

Sau đó đổi chỗ để đưa phần tử bằng v vào giữa



Demo

- demo-partition3.ppt

Code

```
void sort(int a[], int l, int r) {
   if (r <= l) return;
   int i = l-1, j = r;
   int p = l-1, q = r;
   while(1)     {
        while (a[++i] < a[r]);
        while (a[r] < a[--j]) if (j == l) break;
        if (i >= j) break;
        exch(a, i, j);
        if (a[i]==a[r]) exch(a, ++p, i);
        if (a[j]==a[r]) exch(a, --q, j);
   }
   exch(a, i, r);
   j = i - 1;
   i = i + 1;
   for (int k = l ; k <= p; k++) exch(a, k, j--);
   for (int k = r-1; k >= q; k--) exch(a, k, i++);
   sort(a, l, j);
   sort(a, i, r);
}
```

Bài 1

- Viết 2 hàm sắp xếp
  - 2-way partitioning
  - 3-way partitioning
- Tạo 2 mảng gồm 10 triệu số nguyên ngẫu nhiên từ 1-10
- So sánh thời gian sắp xếp của mỗi giải thuật

Gợi ý (1)

- Viết hàm tạo mảng dữ liệu lưu trữ trong bộ nhớ động, kích thước mảng được truyền qua tham số
  - int * createArray(int size);
- Gọi hàm rand() để khởi tạo số ngẫu nhiên trong phạm vi từ 0 tới RAND_MAX
  - #include <stdlib.h>
  - i = rand();
- Hàm cho phép sao chép một mảng số nguyên đã có
  - int * dumpArray(int *p, int size);

Gợi ý (2)

- Hàm memcpy() cho phép sao chép hai vùng nhớ
  - memcpy(void* dest, void* src, size_t size);
- Viết 2 hàm sắp xếp
  - void sort2way(int a[], int l, int r);
  - void sort3way(int a[], int l, int r);
- Trong hàm main(), đầu tiên kiểm tra việc sắp xếp đúng với số lượng phần tử nhỏ. Sau đó tiến hành so sánh thời gian với số lượng phần tử lớn

Gợi ý (3)

```
#define SMALL_NUMBER 20
#define HUGE_NUMBER 10000000
main() {
    int* a1, a2;
    a1 = createArray(SMALL_NUMBER);
    a2 = dumpArray(a1, SMALL_NUMBER);
  sort2way(a1, 0, SMALL_NUMBER-1);
    /* print data in a1 */
    sort2way(a2, 0, SMALL_NUMBER-1);
    /* print data in a2 */
    free (a1);
    free (a2);
    a1 = createArray(HUGE_NUMBER);
    a2 = dumpArray(a1, HUGE_NUMBER);
    /* compare the time to execute sorting */
}
```

Gợi ý (4)

- Kiểm tra thời gian chạy

```
#include <time.h>
#include <stdio.h>
time_t start,end;
volatile long unsigned t;

start = time(NULL);

/* your algorithm to check the performance */

end = time(NULL);

printf(“Run in %f seconds.\n", difftime(end, start));
```

Sắp xếp tổng quát

- Hàm qsort

```
void qsort(
      void *buf,
      size_t num,
      size_t size,
 int (*compare)(void const *,void const *)
);
```

- Hàm qsort, sắp xếp buf (bao gồm num phần tử, mỗi phần tử có kích thước size)
- Hàm compare được sử dụng để so sánh 2 phần tử trong buf. Hàm này nên trả về
  - 0: giống nhau
  - >0: tham số thứ nhất lớn hơn
  - <0: tham số thứ nhất nhỏ hơn.

Ví dụ

```
int int_compare(void const* x, void const *y) {
   int m, n;
   m = *((int*)x);
   n = *((int*)y);
   if ( m == n ) return 0;
   return m > n ? 1: -1;
}
void main()
{
   int a[20], n;
   /* input an array of numbers */
   /* call qsort */
   qsort(a, n, sizeof(int), int_compare);
}
```

Con trỏ hàm

- Khai báo một con trỏ hàm
  - int (*pf) (int);
- Khai báo một hàm
  - int f(int);
- Gán một con trỏ hàm tới hàm
  - pf = &f;
- Gọi hàm qua con trỏ
  - ans = pf(5); // which are equivalent with ans = f(5)
- Trong hàm qsort, compare là con trỏ hàm, tham chiếu tới hàm sắp xếp 2 phần tử

Bài 2

- Sử dụng hàm qsort để sắp xếp mảng theo thứ tự tăng dần/giảm dần
- Viết lại Bài 1 để so sánh hiệu quả giải thuật