Ứng dụng sắp xếp trong nâng cao hiệu quả của thuật toán

Trương Phước Hải

Ý nghĩa tính hiệu quả của thuật toán

- Tính hiệu quả của thuật toán được ước lượng bằng số thao tác sơ cấp mà thuật toán thi hành
- Thiết kế thuật toán hiệu quả giúp xây dựng được ứng dụng có thời gian thực thi trong giới hạn cho phép

Ý nghĩa tính hiệu quả của thuật toán

- Phân tích tính hiệu quả của thuật toán
  - Có vai trò khá quan trọng khi độ lớn của dữ liệu tăng lên
  - Giúp phát hiện và loại bỏ các thao tác tính toán dư thừa
  - Đánh giá được tính ứng dụng của thuật toán trong từng lĩnh vực cụ thể

Sắp xếp và ứng dụng

- Sắp xếp là quá trình bố trí lại các phần tử trong dãy theo một thứ tự xác định
- Sắp xếp là thao tác tiền xử lý dữ liệu giúp nâng cao hiệu quả các bước xử lý tiếp theo
- Một số chiến lược tìm kiếm hiệu quả phần lớn dựa trên tính có thứ tự của dữ liệu

Sắp xếp và ứng dụng

- Một số chiến lược tìm kiếm phổ dụng
  - Tìm kiếm tuần tự: Xét toàn bộ không gian phần tử để tìm phần tử thỏa yêu cầu
  - Tìm kiếm nhị phân: Dựa trên tính có thứ tự của không gian phần tử để loại bỏ vùng chắc chắn không chứa phần tử cần tìm

Bài toán

- Cho dãy số nguyên $a_0, a_1, \dots, a_{n-1}$. Đếm số cặp vị trí (i, j) sao cho $a_i + a_j = x (i < j)$, với x cho trước

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
2	5	13	7	5	8	5	8	-3	5	7	-3	8	8	2

10
x

Cách 1: phương pháp tầm thường (vét cạn)

- Xét tất cả cặp vị trí $i, j (i < j)$ và kiểm tra $a_i + a_j = x$

```
ans = 0;
for (i = 0; i < n; ++i)
    for (j = i+1; j < n; ++j)
        if (a[i] + a[j] == x)
            ++ans;
```

- Độ phức tạp $O(n^2)$

Cách 2: tìm kiếm nhị phân (cải tiến cách 1)

- Nhận xét $a_i + a_j = x \Longrightarrow a_j = x - a_i$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
2	5	13	7	5	8	5	8	-3	5	7	-3	8	8	2	10
a_i															x

- Với mỗi giá trị a_i, đếm trong dãy số lượng phần tử a_j thỏa $a_j = x - a_i (i \neq j)$

Cách 2: tìm kiếm nhị phân (cải tiến cách 1)

- Bản chất của bài toán là tìm kiếm
  - Thuật toán tìm kiếm hiệu quả: tìm kiếm nhị phân
  - Điều kiện: dãy phải có thứ tự (tăng dần)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
2	5	13	7	5	8	5	8	-3	5	7	-3	8	8	2

10

x

a_i

lower bound

upper bound

Cách 2: tìm kiếm nhị phân (cải tiến cách 1)

```
ans = 0;
for (i = 0; i < n; ++i)
{
    low = lower(x – a[i], i+1, n-1);
    upp = upper(x – a[i], i+1, n-1);
    //nếu dãy chứa phần tử có giá trị x – a[i]
    if (low >= 0 && upp >= 0)
        ans = ans + (upp – low + 1);
}
```

Cách 2: tìm kiếm nhị phân (cải tiến cách 1)

- Độ phức tạp của thuật toán phụ thuộc vào hàm `lower` và hàm `upper`.
- Do dãy có thứ tự (tăng dần) nên có thể áp dụng kỹ thuật chặt nhị phân
  - `lower(x, lo, hi)`: vị trí nhỏ nhất từ `lo` đến `hi` có giá trị x.
  - `upper(x, lo, hi)`: vị trí lớn nhất từ `lo` đến `hi` có giá trị x.

Cách 2: tìm kiếm nhị phân (cải tiến cách 1)

```
lower(x, lo, hi) {
    pos = -1;
    while (lo <= hi) {
        m = (lo + hi)/2;
        if (a[m] >= x) {
            if (a[m] == x) pos = m;
            hi = m-1;
        }
        else lo = m+1;
    }
    return pos;
}
```

Cách 2: tìm kiếm nhị phân (cải tiến cách 1)

```
upper(x, lo, hi) {
    pos = -1;
    while (lo <= hi) {
        m = (lo + hi)/2;
        if (a[m] <= x) {
            if (a[m] == x) pos = m;
            lo = m+1;
        }
        else hi = m-1;
    }
    return pos;
}
```

Cách 2: tìm kiếm nhị phân (cải tiến cách 1)

- Đánh giá thuật toán cách 2
  - Độ phức tạp của `lower` và `upper` đều là $O(\log n)$
  - Độ phức tạp của thuật toán: $O(n \log n)$

Cách 3: nén và thống kê dãy (cải tiến cách 2)

- Cách 2 thực hiện 2 lần chặt nhị phân để tìm lower và upper của đoạn giá trị $x - a_i$
- Giảm số lần chặt nhị phân: nén các đoạn bằng nhau thành các giá trị phân biệt cùng tần số xuất hiện

Cách 3: nén và thống kê dãy (cải tiến cách 2)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$a[0..n-1]$	-3	-3	2	2	5	5	5	5	7	7	8	8	8	8	13

Nén dãy và thống kê

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$val[0..m-1]$	-3	-3	2	2	5	5	5	5	7	7	8	8	8	8	13

$frq[0..m-1]$	2	2	4	2	4	1

Cách 3: nén và thống kê dãy (cải tiến cách 2)

- Nếu tồn tại $i, j (i < j)$ thỏa $val[i] + val[j] = x$ thì số cặp có tổng bằng x là $frq[i] * frq[j]$
- Trường hợp đặc biệt: $x = 2 \times val[i]$, số cặp có tổng bằng x là $C_{frq[i]}^{2} = frq[i] \times (frq[i] - 1)/2$

Cách 3: nén và thống kê dãy (cải tiến cách 2)

```
ans = 0;
for (i = 0; i < m; ++i)
{
    if (x == 2*val[i])
        ans = ans + frq[i]*(frq[i] - 1)/2;
    else
    {
        j = BinarySearch(x – val[i], i + 1, m-1);
        if (j >= 0) ans = ans + frq[i]*frq[j];
    }
}
```

Cách 3: nén và thống kê dãy (cải tiến cách 2)

- Phương pháp nén và thống kê thực hiện các công đoạn
  - Công đoạn 1: sắp xếp dãy, độ phức tạp $O(n \log n)$
  - Công đoạn 2: nén và thống kê dãy, độ phức tạp $O(n)$
  - Công đoạn 3: đếm số cặp, độ phức tạp $O(m \log m)$, với m là số giá trị phân biệt của dãy

Cách 3: kỹ thuật 2 con trỏ (một hướng nhìn khác)

- Nhận xét công đoạn 3
  - Nếu $val[i] + val[j] = x$, khi đó $val[i]$ tăng thì $val[j]$ giảm
  - Dãy có thứ tự tăng dần nên i và j thay đổi ngược nhau

	$\overset{i}{\rightarrow}$					$\overset{j}{\leftarrow}$
$val[0..m-1]$	-3	2	5	7	8	13
$frq[0..m-1]$	2	2	4	2	4	1

Cách 3: kỹ thuật 2 con trỏ (một hướng nhìn khác)

```
ans = 0; i = 0, j = m-1;
while (i <= j) {
    if (val[i] + val[j] == x) {
        if (val[i] == val[j])
            ans = ans + frq[i]*(frq[i] – 1)/2;
        else ans = ans + frq[i]*frq[j];
        ++i, --j;
    }
    else if (val[i] + val[j] < x) ++i;
    else --j;
}
```

Cách 3: kỹ thuật 2 con trỏ (một hướng nhìn khác)

- Đánh giá cách 3 dùng kỹ thuật 2 con trỏ
  - Công đoạn 1: sắp xếp dãy, độ phức tạp $O(n \log n)$
  - Công đoạn 2: nén và thống kê dãy, độ phức tạp $O(n)$
  - Công đoạn 3: đếm số cặp bằng kỹ thuật 2 con trỏ, độ phức tạp $O(m)$, với m là số giá trị phân biệt của dãy

Cách 4: kỹ thuật mảng đánh dấu (cải tiến từ cách 3)

- Nếu miền giá trị của a_i đủ nhỏ $(0 \leq a_i \leq 10^6)$ thì sử dụng kỹ thuật mảng thống kê

frq[v] = 0, $\forall v \in [\min\{a_i\}..\max\{a_i\}]$

```
for (i = 0; i < n; ++i)
    ++frq[a[i]];
```

Cách 4: kỹ thuật mảng đánh dấu (cải tiến từ cách 3)

- Mảng thống kê frq giúp tìm $v = a[i]$ trong $O(1)$ thay vì tìm kiếm nhị phân trong $O(\log n)$

```
ans = 0;
for (v = min{a[i]}; v <= x - v; ++v) {
    if (x - v <= max{a[i]}) {
        if (v == x - v)
            ans = ans + frq[v]*(frq[v]-1)/2;
        else ans = ans + frq[v]*frq[x - v];
    }
}
```

Cách 4: kỹ thuật mảng đánh dấu (cải tiến từ cách 3)

- Thuật toán cách 4 thực hiện qua 2 công đoạn
  - Công đoạn 1: thống kê mảng, độ phức tạp $O(n)$
  - Công đoạn 2: duyệt trên miền giá trị của a_i để đếm số cặp thỏa yêu cầu, độ phức tạp $O(\max\{a_i\})$

Cách 4: kỹ thuật mảng đánh dấu (cải tiến từ cách 3)

- Nhận xét
  - Thông tin thống kê của các giá trị $v = a[i]$ được lưu trữ tại vị trí v trong mảng `frq`
  - Kích thước mảng `frq` tương ứng kích thước miền giá trị $a[i]$
  - Giải pháp cho trường hợp $a[i] < 0$?

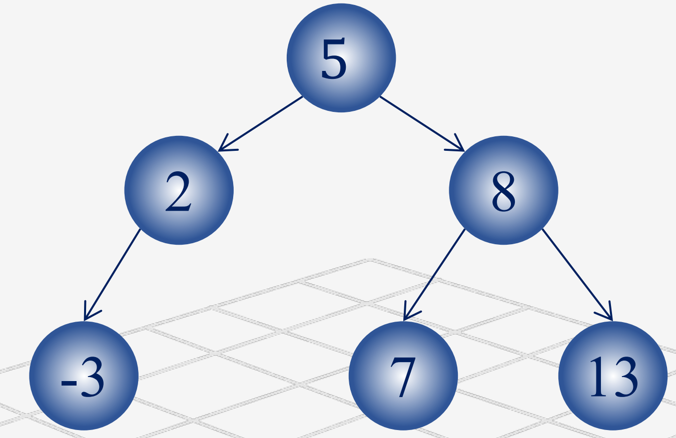
Cách 5: sử dụng cấu trúc dữ liệu

- Kết hợp ý tưởng cách 3 và cách 4. Tổ chức cây nhị phân tìm kiếm tự cân bằng. Mỗi node của cây lưu thông tin
  - Trường giá trị phân biệt
  - Trường tần số xuất hiện
- Thao tác thêm/tìm kiếm một node trên cây có độ phức tạp gần với $O(\log m)$, với m là số giá trị phân biệt

Cách 5: sử dụng cấu trúc dữ liệu

- Sử dụng cây nhị phân tìm kiếm tự cân bằng

2	5	13	7	5	8	5	8	-3	5	7	-3	8	8	2



Cách 5: sử dụng cấu trúc dữ liệu

- Cây nhị phân tìm kiếm tự cân bằng có thể sử dụng cây AVL hoặc cây đỏ đen
- Thư viện standard của C++ cung cấp cấu trúc `map` và `multiset` có nền tảng là cây nhị phân tự cân bằng
- Mỗi phần tử của `map/multiset` ánh xạ cặp giá trị `(key, value)`, trong đó key là các giá trị phân biệt

Cách 5: sử dụng cấu trúc dữ liệu

- Khai báo và tổ chức dãy số nguyên $a_0, a_1, \dots, a_{n-1}$ thành một map

```
map<key_type, value_type> m;
for (i = 0; i < n; ++i)
    ++m[a[i]];
```

Cách 5: sử dụng cấu trúc dữ liệu

- Duyệt map để đếm số cặp có tổng là x

```
ans = 0;
map<key_type, value_type>::iterator it;
for (it = m.begin(); it != m.end(); ++it)
{
    v = it->first, u = x - v;
    if (v < u) ans = ans + m[v]*m[u];
    else if (v == u)
        ans = ans + m[v]*(m[v] - 1)/2;
}
```

Cách 5: sử dụng cấu trúc dữ liệu

- Thuật toán cách 5 thực hiện qua 2 công đoạn
  - Công đoạn 1: xây dựng dãy thành một `map` (cây nhị phân tìm kiếm tự cân bằng), độ phức tạp cỡ $O(n \log m)$
  - Công đoạn 2: tìm từng cặp khóa trong `map` có tổng bằng x, độ phức tạp cỡ $O(m \log m)$, với m là số khóa trên `map`

Bài tập áp dụng

- Bài 1: Cho dãy số $a_0, a_1, \dots, a_{n-1}$. Đếm số cặp phần tử $a_i, a_j (i < j)$ thỏa $a_i + a_j \leq x$
- Bài 2: Cho dãy số $a_0, a_1, \dots, a_{n-1}$. Xét tất cả các tổng $a_i + a_j (i < j)$ theo thứ tự tăng dần. Tìm giá trị tổng thứ k
- Bài 3: Cho dãy số $a_0, a_1, \dots, a_{n-1}$. Tìm 3 vị trí i, j, k thỏa $a_i + a_j + a_k = x (0 \leq i < j < k \leq n - 1)$

Bài tập áp dụng

- Bài 4: Cho dãy số $a_0, a_1, \ldots, a_{n-1}$. Tìm 4 vị trí i, j, k, t thỏa $a_i + a_j + a_k + a_t = x (0 \le i < j < k < t \le n-1)$
- Bài 5: Cho dãy số $a_0, a_1, \ldots, a_{n-1}$. Đếm số đoạn con (gồm các phần tử liên tiếp nhau) của dãy có tổng đúng bằng x
- Bài 6: Cho dãy số $a_0, a_1, \ldots, a_{n-1}$. Các phần tử được nhóm thành các dãy giảm dần. Đếm số nhóm ít nhất tạo được

Bài tập áp dụng

- Bài 7: Cho 2 dãy số $a_0, a_1, \dots, a_{n-1}$ và $b_0, b_1, \dots, b_{m-1}$. Đếm số cặp a_i, b_j thỏa
  - $a_i < b_j$
  - $a_i \neq b_j$
- Bài 8: Cho 2 dãy số $a_0, a_1, \dots, a_{n-1}$ và $b_0, b_1, \dots, b_{m-1}$. Đếm số cặp phần tử (a_i, b_j) thỏa $a_i + b_j < s$

Bài tập áp dụng

- Bài 9: Cho 2 dãy số $a_0, a_1, \dots, a_{n-1}$ và $b_0, b_1, \dots, b_{m-1}$. Tìm giá trị lớn nhất của $a_i + b_j$ thỏa $a_i + b_j < s$
- Bài 10: Cho 2 dãy số $a_0, a_1, \dots, a_{n-1}$ và $b_0, b_1, \dots, b_{m-1}$. Đếm số cặp $\left(a_i, b_j\right)$ thỏa $a_i + b_j = x$
- Bài 11: Cho 2 dãy số $a_0, a_1, \dots, a_{n-1}$ và $b_0, b_1, \dots, b_{m-1}$. Tìm các giá trị xuất hiện ở cả 2 dãy

Bài tập áp dụng

- Bài 12: Cho 2 dãy số $a_0, a_1, \ldots, a_{n-1}$ và $b_0, b_1, \ldots, b_{m-1}$. Đếm số cặp $\left(a_i, b_j\right)$ thỏa $a_i < b_j$
- Bài 13: Cho 2 dãy số $a_0, a_1, \ldots, a_{n-1}$ và $b_0, b_1, \ldots, b_{m-1}$. Tìm giá trị nhỏ nhất của $\left|a_i - b_j\right|$
- Bài 14: Cho 2 dãy số $a_0, a_1, \ldots, a_{n-1}$ và $b_0, b_1, \ldots, b_{m-1}$. Tìm giá trị nhỏ nhất của $\left|a_i + b_j\right|$