NGUYEN XUAN HUY

ARRAY SORTING IN C++

Hà Nội – 2021

MỤC LỤC

Array Sorting

Sort theo tiêu chí định sẵn

Đôi khi ta gọi kiểu sort này là sắp xếp tự nhiên.

Trong C++ để *sắp tăng* một đoạn trong mảng một chiều a từ phần tử a[i] đến phần tử a[j], bạn chỉ cần gọi

```
sort(a+i, a+j+1);
```

Muốn sắp tăng toàn bộ n phần tử của mảng a, bạn gọi

```
sort(a, a+n);
```

Chương trình minh hoạ C++

```
// ArraySort.CPP
#include <iostream>
#include <bits/stdc++.h>
using namespace std;

const int N = 10;
int a[] = {2, 6, 0, 8, 1, 7, 4, 9, 3, 5};
int b[N];

// Hien thi mang 1D a
void Print(int a[], int i1, int i2, char * msg = "") {
  cout << msg;
  for (int i = i1; i <= i2; ++i) {
    cout << " " << a[i];
  }
}

int main() {
   memcpy(b,a, sizeof(a));
   Print(a, 0, N-1, "\n Given a: ");
   sort(a, a+N);
   Print(a, 0, N-1, "\n Sorted a: ");
   Print(b, 0, N-1, "\n Given b: ");
   sort(b+1, b+N);
   Print(b, 0, N-1, "\n Sorted b[1:9]: ");
   cout << "\n T h e    E n d .\n";
   return 0;
}
```

Kết quả

```
 Given a:  2 6 0 8 1 7 4 9 3 5
 Sorted a:  0 1 2 3 4 5 6 7 8 9
 Given b:  2 6 0 8 1 7 4 9 3 5
 Sorted b[1:9]:  2 0 1 3 4 5 6 7 8 9
 T h e    E n d .
```

Bạn muốn sắp giảm mảng số thực a[i:j], bạn viết

```
sort(a+i, a + j + 1, greater<float>());
```

Như vậy, hàm greater<float>() sẽ so sánh hai đối tượng kiểu float trong a[] theo chiều giảm.

Chương trình minh hoạ C++

```
// ArraySort.CPP
#include <iostream>
#include <bits/stdc++.h>
using namespace std;

const int N = 10;
float a[] = {2, 6, 0, 8, 1, 7, 4, 9, 3, 5};
float b[N];

void Print(float a[], int i1, int i2, char * msg = "") {
```

```
  cout << msg;
  for (int i = i1; i <= i2; ++i) {
    cout << " " << a[i];
  }
}

int main() {
   memcpy(b, a, sizeof(a));
   Print(a, 0, N-1, "\n  Given a: ");
   sort(a, a+N, greater<float>());
   Print(a, 0, N-1, "\n dec sorted a: ");
   Print(b, 0, N-1, "\n Given b: ");
   sort(b+3, b+7, greater<float>());
   Print(b, 0, N-1, "\n dec Sorted b[3:6]: ");
   cout << "\n\n T h e    E n d .\n";
   return 0;
}
```

Kết quả

```
  Given a:  2 6 0 8 1 7 4 9 3 5
  dec sorted a:  9 8 7 6 5 4 3 2 1 0
  Given b:  2 6 0 8 1 7 4 9 3 5
  dec Sorted b[3:6]:  2 6 0 8 7 4 1 9 3 5
  T h e    E n d .
```

Sort theo tiêu chí tự đặt

Đôi khi bạn muốn thay đổi tiêu chí so sánh các phần tử của mảng, ví dụ bạn muốn sắp tăng mảng nguyên không âm a theo độ cao H của mỗi số.

Độ cao H của một số nguyên không âm *x* là tổng các chữ số của số *x*.

Ví dụ

```
H(123) = 1 + 2 + 3 = 6, H(10) = 1 + 0 = 1, H(0) = 0.
```

Bạn muốn cài đặt hàm H theo tùy biến, cụ thể là H(x) sẽ cho ra tổng các chữ số của số nguyên x theo hệ đếm 10 (hệ thập phân), còn H(x, 2) sẽ cho ra tổng các chữ số của số nguyên x theo hệ đếm 2. Nói cách khác , H(x, 2) cho biết trong dạng nhị phân biểu diễn số nguyên x có bao nhiêu bit 1.

Ví dụ

```
H(19) = 1 + 9 = 10, H(19, 2) = 3,vì 19₂ = 10011.
```

Việc cài đặt hàm H(x) khá dễ, bạn chỉ việc lấy tổng các chữ số của x là x mod base tính theo hệ đếm nào thì ta chỉ việc chia theo hệ đếm đó. Hàm x div base bỏ đi chữ số đơn vị của x.

Hàm H(x)

```
int H(int x, base = 10) {
  int digit_sum = 0;
  while (x != 0) {
    digit_sum += x % base;
    x /= base;
  }
  return digit_sum;
}
```

Tiếp đến bạn phải tự cài đặt phương thức so sánh hai phần tử của mảng a. Bạn quy định rằng

x < y khi và chỉ khi H(x) < H(y):

```
bool Less(int x, int y) {
  return H(x) < H(y);
}
```

Đến đây bạn gọi hàm sort:

```
sort(a, a+n, Less);
```

Nếu muốn sắp giảm bạn gọi

```
sort(a, a+n, Great);
```

với hàm Great do bạn tự viết như sau:

```
bool Great(int x, int y) {
  return H(x) > H(y);
}
```

Chương trình minh hoạ C++

```
// ArraySort.CPP
#include <iostream>
#include <bits/stdc++.h>
using namespace std;

const int N = 10;
int a[] = {20, 16, 10, 0, 111, 73, 46, 19, 33, 15};
int b[N];

// Do cao theo base 10 | 2
int H(int x, int base = 10) {
  int digit_sum = 0;
  while (x != 0) {
    digit_sum += x % base;
    x /= base;
  }
  return digit_sum;
}

// Hien thi a[i1]...a[i2] va do cao theo base 10 | 2
void Print(int a[], int i1, int i2, char * msg = "", int base = 10) {
  cout << msg;
  for (int i = i1; i <= i2; ++i) {
    cout << " " << a[i] << ":" << H(a[i], base);
  }
}

// x < y theo base 10
bool Less(int x, int y) {
  return H(x) < H(y);
}

// x > y theo base 10
bool Great(int x, int y) {
  return H(x) > H(y);
}

// x < 2 theo base 2
bool Less2(int x, int y) {
  return H(x, 2) < H(y, 2);
}

// x > y theo base 2
bool Great2(int x, int y) {
  return H(x, 2) > H(y, 2);
}

int main() {
   memcpy(b, a, sizeof(a));
   // b = a
   Print(a, 0, N-1, "\n  Given a in base 10: ");
   sort(a, a+N, Less);
   Print(a, 0, N-1, "\n inc sorted a by H of base 10: ");
   sort(a, a+N, Great);
   Print(a, 0, N-1, "\n dec sorted a by H of base 10: ");
   Print(a, 0, N-1, "\n Now a in base 2: ", 2);
   sort(a, a+N, Less2);
   Print(a, 0, N-1, "\n inc sorted a by H of base 2: ", 2);
   sort(a, a+N, Great2);
```

```
   Print(a, 0, N-1, "\n dec sorted a by H of base 2: ", 2);

   int i1 = 2, i2 = 7;
   Print(b, 0, N-1, "\n  Given b in base 10: ");
   sort(b+i1, b+i2+1, Less);
   Print(b, 0, N-1, "\n inc sorted b[2:7] by H of base 10: ");
   sort(b+i1, b+i2+1, Great);
   Print(b, 0, N-1, "\n dec sorted b[2:7] by H of base 10: ");
   Print(b, 0, N-1, "\n Now b in base 2: ", 2);
   sort(b+i1, b+i2+1, Less2);
   Print(b, 0, N-1, "\n inc sorted b[2:7] by H of base 2: ", 2);
   sort(b+i1, b+i2+1, Great2);
   Print(b, 0, N-1, "\n dec sorted b[2:7] by H of base 2: ", 2);
   cout << "\n T h e    E n d .\n";
   return 0;
}
```

Kết quả

```
Given a in base 10:  20:2 16:7 10:1 0:0 111:3 73:10 46:10 19:10 33:6 15:6
inc sorted a by H of base 10: 0:0 10:1 20:2 111:3 33:6 15:6 16:7 73:10 46:10 19:10
dec sorted a by H of base 10: 73:10 46:10 19:10 16:7 33:6 15:6 111:3 20:2 10:1 0:0
Now a in base 2:  73:3 46:4 19:3 16:1 33:2 15:4 111:6 20:2 10:2 0:0
inc sorted a by H of base 2: 0:0 16:1 33:2 20:2 10:2 73:3 19:3 46:4 15:4 111:6
dec sorted a by H of base 2: 111:6 46:4 15:4 73:3 19:3 33:2 20:2 10:2 16:1 0:0
Given b in base 10:  20:2 16:7 10:1 0:0 111:3 73:10 46:10 19:10 33:6 15:6
inc sorted b[2:7] by H of base 10: 20:2 16:7 0:0 10:1 111:3 73:10 46:10 19:10 33:6 15:6
dec sorted b[2:7] by H of base 10: 20:2 16:7 73:10 46:10 19:10 111:3 10:1 0:0 33:6 15:6
Now b in base 2:  20:2 16:1 73:3 46:4 19:3 111:6 10:2 0:0 33:2 15:4
inc sorted b[2:7] by H of base 2: 20:2 16:1 [0:0 10:2 73:3 19:3 46:4 111:6] 33:2 15:4
dec sorted b[2:7] by H of base 2: 20:2 16:1 111:6 46:4 73:3 19:3 10:2 0:0 33:2 15:4
T h e    E n d .
```

Sort theo chỉ dẫn

Trong quá trình sort, các đối tượng trong mảng thường phải đổi chỗ với nhau cho nên chi phí cho các phép toán copy thường làm tăng thời gian sắp xếp, đặc biệt là với những đối tượng có kích thước lớn như string hoặc bản ghi. Sort theo chỉ dẫn cho phép chúng ta chỉ ra *một trật tự*, ví dụ, tăng dần, các đối tượng của mảng nhưng vẫn phần *giữ nguyên vị trí ban đầu* của chúng.

Ví dụ

Ta có mảng name chứa danh sách các bạn trẻ. Ta cần duyệt danh sách này theo trật tự từ điển như dòng 4 trong bảng dưới đây:

init name	HOA	KHANH	AN	TUAN	BINH	VINH	CHINH	VAN	DUNG	MAI
Init id	0	1	2	3	4	5	6	7	8	9
sorted id	2	4	6	8	0	1	9	3	7	5
Print name by id	AN	BINH	CHINH	DUNG	HOA	KHANH	MAI	TUAN	VAN	VINH

Để thực hiện điều này ta cần một mảng phụ tạm gọi là id để quản lý số thứ tự của mảng a.

Quy trình sort sẽ được thực hiện qua các bước sau:

Bước 1. Khởi trị cho mảng id:

```
id[i] = i, 0 ≤ i < n
```

trong đó n là số phần tử của mảng a.

Dòng 2 của bảng cho ta kết quả của bước khởi trị id.

Bước 2. Cài đặt hàm Less(i,j) chỉ rõ phương thức so sánh hai đối tượng name[i] và name[j], cụ thể là cho biết name[i] < name[j] ?

```
bool Less(int i, int j) {
  return name[i] < name[j];
}
```

Bước 3. Gọi hàm sort mảng id chứ không phải mảng name:

```
sort(id, id+n, Less);
```

với ý nghĩa sắp xếp lại mảng phụ id từ vị trí đầu đến cuối theo phương thức so sánh Less.

Như vậy chỉ có *các phần tử trong id được sắp xếp lại*, còn bản thân các string trong mảng name vẫn được giữ nguyên.

Dòng 3 của bảng cho ta kết quả của bước 3

```
id = [2, 4, 6, 8, 0, 1, 9, 3, 7, 5]
```

với ý nghĩa:

```
id[0] = 2 cho biết phần tử name[2] (AN) sẽ đứng ở vị trí đầu tiên,
id[1] = 4 cho biết phần tử name[4] (BINH) sẽ đứng ở vị trí thứ hai...
```

Ngoài ưu thế tiết kiệm thời gian, sort theo chỉ dẫn còn được ứng dụng trong các bài toán cần tham chiếu đến vị trí ban đầu của dữ liệu.

Chương trình minh hoạ C++

```
// ArraySort.CPP
#include <iostream>
#include <bits/stdc++.h>
using namespace std;

string name[] = {"HOA", "KHANH", "AN", "TUAN", "BINH",
                 "VINH", "CHINH", "VAN", "DUNG", "MAI"};
int n = 10;
int *id;

void Print(string a[], int n, char * msg = "") {
  cout << msg;
  for (int i = 0; i < n; ++i) {
    cout << " " << a[i];
  }
}

void Print(int x[], int n, char * msg = "") {
  cout << msg;
  for (int i = 0; i < n; ++i) {
    cout << " " << x[i];
  }
}

void PrintById(string *a, int *id, int n, char * msg = "") {
  cout << msg;
  for (int i = 0; i < n; ++i) {
    cout << " " << a[id[i]];
  }
}

bool Less(int i, int j) {
  return name[i] < name[j];
}

int main() {
   Print(name, n, "\n  Given a: ");
   id = new int[n];
   for (int i = 0; i < n; ++i) {
     id[i] = i;
   }
   Print(id, n, "\n  Init id: ");
   sort(id, id+n, Less);
   Print(id, n, "\n  Now id: ");
   PrintById(name, id, n, "\n Now Names: ");
```

```
    cout << "\n T h e    E n d .\n";
    return 0;
}
```

Kết quả

```
Given a:  HOA KHANH AN TUAN BINH VINH CHINH VAN DUNG MAI
Init id:  0 1 2 3 4 5 6 7 8 9
Now id:  2 4 6 8 0 1 9 3 7 5
Now Names:  AN BINH CHINH DUNG HOA KHANH MAI TUAN VAN VINH
T h e    E n d .
```

Tổng kết các hàm Sort trong C++

sort 1D array a[i:j]

sort(a+i, a+j+1)	*sắp tăng*
sort(a+i, a+j+1, Less)	*sắp tăng theo tiêu chí tự đặt*
sort(a+i, a+j+1, greater<type>())	*sắp giảm*
sort(a+i, a+j+1, Great)	*sắp giảm theo tiêu chí tự đặt*

type: int, float, string,..., Less, Great: tự đặt

ID sort 1D array a[i:j]

id[i] = i, $0 \le i < n$

sort(id+i, id+j, Less)	*sắp tăng*
sort(id+i, id+j, Great)	*sắp giảm*

Less, Great tự đặt

Tự viết các hàm Sort

Method:

1. Lấy giá trị m tại điểm giữa mảng.
2. Chuyển các a[i] < m về nửa trái, a[j] > m về nửa phải.
3. Sắp tăng nửa trái
4. Sắp tăng nửa phải

Program

```
#include <iostream>
#include <algorithm>
#include <time.h>

using namespace std;

const int MN = 300000;

int a[MN] = {5, 2, -1, 9, 6, -7, 12, 19, -14, 40};

void Go(char * msg = " ? ") {
   cout << msg;
   fflush(stdin);
   if (cin.get() == '.') exit(0);
}

void Print(int a[], int d, int c, const char * msg = "") {
  cout << msg;
  for (int i = d; i <= c; ++i)
     cout << " " << a[i];
}

void Swap(int &a, int &b) {
 int t = a; a = b; b = t;
}

// O(n^2)
void MinSort(int a[], int d, int c) {
   for (int t = d; t <= c; ++t) {
      for (int p = t + 1; p <= c; ++p) {
         if (a[p] < a[t]) Swap(a[p], a[t]);
      }
   }
}

// O(n^2)
void BubbleSort(int a[], int d, int c) {
   for (int t = d; t <= c; ++t) {
      for (int p = c; p > t; --p) {
         if (a[p] < a[p-1]) Swap(a[p], a[p-1]);
      }
   }
}

// O(nlog(n))
void QuickSort(int a[], int d, int c) {
  int t = d, p = c;
   int m = a[(t+p) / 2];
   while (t <= p) {
      while(a[t] < m) ++t;
      while(a[p] > m) --p;
      if (t <= p) {
         Swap(a[t], a[p]);
         ++t; --p;
      }
   }
```

```
   if (d < p) QuickSort(a, d, p);
  if (t < c) QuickSort(a, t, c);
}

void Test1() {
  int n = 10;
  Print(a, 0, n-1,"\n Given a: ");
  // MinSort(a, 0, n-1);
  // BubbleSort(a, 0, n-1);
  QuickSort(a, 0, n-1);
  Print(a, 0, n-1,"\n Sorted a: ");
}


void Gen(int a[], int n) {
   srand(time(NULL));
   for (int i = 0; i < n; ++i)
       a[i] = rand() % 10000;
}

void Test2() {
  int n = 30000;
  Gen(a, n);
  // Print(a, 0, 200, "\n Init: ");
  int t = time(NULL);
  // MinSort(a, 0, n-1);
  // BubbleSort(a, 0, n-1);
  QuickSort(a, 0, n-1);
  cout << "\n End of QuickSort, time = " << difftime(time(NULL),t);
  // cout << "\n End of MinSort, time = " << difftime(time(NULL),t);
  //cout << "\n End of BubbleSort, time = " << difftime(time(NULL),t);
}

main() {
  // Test1();
  Test2();
  cout << "\n T h e   E n d";
  return 0;
}
```

Độ phức tạp tính toán

MinSort and BubbleSort: O(n2); QuickSort: O(nlog(n)).

nxhuy564@gmail.com