

#04-3. 정렬 STL

나정휘

<https://justicehui.github.io/>

목차

비교 기반 정렬

Strict Weak Ordering

정렬 STL

비교 기반 정렬

비교 기반 정렬

- 두 원소를 비교/교환해서 정렬하는 알고리즘
- ex. 오름차순 정렬
 - $x < y$ and $A[y] < A[x]$ 인 (x, y) 가 없어야 함
 - $x < y$ and $A[y] < A[x]$ 이면 $A[x]$ 와 $A[y]$ 교환

비교 기반 정렬

비교 함수

- `bool Compare(T a, T b)`
 - a가 b보다 반드시 앞에 나와야 하면 `true`, 아니면 `false`
 - $x < y$ and `Compare(A[y], A[x]) = true` 인 (x, y) 가 없어야 함
 - $x < y$ and `Compare(A[y], A[x]) = true` 이면 $A[x]$ 와 $A[y]$ 교환
 - **Strict Weak Ordering**을 만족해야 함
 - 오름차순 정렬: `Compare(a, b) = a < b` $a \leq b$ 아님
 - 내림차순 정렬: `Compare(a, b) = a > b` $a \geq b$ 아님
- C++에는 비교 함수를 이용해 정렬하는 `std::sort`, `std::stable_sort` 라는 함수가 있음

Strict Weak Ordering

Strict Weak Ordering

- 비반사성(irreflexivity)
 - 모든 x 에 대해 $R(x, x)$ 는 거짓
- 비대칭성(asymmetry)
 - 모든 x, y 에 대해 $R(x, y)$ 이 참이면 $R(y, x)$ 는 거짓
- 추이성(transitivity)
 - 모든 x, y, z 에 대해 $R(x, y), R(y, z)$ 가 참이면 $R(x, z)$ 는 참
- 비비교성의 추이성(transitivity of incomparability)
 - 모든 x, y, z 에 대해 $R(x, y), R(y, x), R(y, z), R(z, y)$ 가 거짓이면 $R(x, z), R(z, x)$ 는 거짓

Strict Weak Ordering

비교성(comparability)

- 비교를 할 수 있다
- (정렬에서) 두 원소의 순서를 정할 수 있다

비비교성(incomparability)

- 비교를 할 수 없다
- (정렬에서) 두 원소의 순서를 정할 수 없다 = 동등하다

ex) 오름차순 정렬

- 값이 같은 두 원소는 순서를 정할 수 없음
- 값이 다른 두 원소는 순서를 정할 수 있음

Strict Weak Ordering

비반사성(irreflexivity)

- 모든 x 에 대해 $R(x, x)$ 는 거짓
- 값이 같은 두 원소는 비교가 불가능하다
- $\text{Compare}(x, x)$ 는 두 x 중 반드시 먼저 와야 하는 것을 결정할 수 없음
- 오름차순의 비교 함수로 $a \leq b$ 를 사용할 수 없는 이유

Strict Weak Ordering

비대칭성(asymmetry)

- 모든 x, y 에 대해 $R(x, y)$ 이 참이면 $R(y, x)$ 는 거짓
 - 두 개가 모두 참이면 두 원소의 순서를 정할 수 없음
 - $R(x, y), R(y, x)$ 모두 거짓이 되어야 함
-
- 사이클이 있는 그래프에서 위상 정렬이 불가능한 이유

Strict Weak Ordering

추이성(transitivity)

- 모든 x, y, z 에 대해 $R(x, y)$, $R(y, z)$ 가 참이면 $R(x, z)$ 는 참
- 삼단 논법

비비교성의 추이성(transitivity of incomparability)

- 모든 x, y, z 에 대해 $R(x, y)$, $R(y, x)$, $R(y, z)$, $R(z, y)$ 가 거짓이면 $R(x, z)$, $R(z, x)$ 는 거짓
- x y 가 동등하고(비교가 불가능하고), y z 가 동등하면 x z 도 동등해야 함

질문?

정렬 STL

std::sort

- sort(first, last): 시작 주소, 끝 주소
 - 기본적으로 오름차순 정렬
 - 구조체/클래스의 경우 < 연산이 정의되어 있어야 함
- sort(first, last, comp): 시작 주소, 끝 주소, 비교 함수
- $O(N \log N)$

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

bool Compare(int a, int b){
    return a > b;
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    int A[5] = {5, 3, 1, 4, 2};

    // 정렬할 범위의 첫 주소, 마지막 주소의 한 칸 뒤 (반열린 구간)
    sort(A, A+5); // 비교 함수 지정 안 하면 오름차순 정렬
    for(int i=0; i<5; i++) cout << A[i] << " \n"[i+1==5];

    sort(A, A+5, Compare); // 비교 함수 사용, 내림차순
    for(int i=0; i<5; i++) cout << A[i] << " \n"[i+1==5];

    // 람다식으로 비교 함수 구현, 오름차순
    sort(A, A+5, [](int a, int b){ return a < b; });
    for(int i=0; i<5; i++) cout << A[i] << " \n"[i+1==5];
}
```

정렬 STL

std::stable_sort

- stable sort와 unstable sort
 - stable sort: 동등한(비교 불가능한) 원소는 기존 순서 유지
 - unstable sort: 동등한 원소들의 순서는 마음대로
- std::sort는 unstable sort
- stable sort를 사용하고 싶다면 std::stable_sort 사용

```
● ● ●

#include <bits/stdc++.h>
using namespace std;

struct Point{
    int x, y;
};

bool Compare(Point a, Point b){
    return a.x < b.x;
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    Point A[3] = { {1, 3}, {1, 2}, {0, 1} };
    stable_sort(A, A+3, Compare);
    // x 좌표가 같은 점들은 기존 순서 유지
    for(int i=0; i<3; i++) cout << A[i].x << " " << A[i].y << "\n";
}
```

질문?