SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

삼성 청년 SW 아카데미

알고리즘

<알림>

본 강의는 삼성 청년 SW아카데미의 컨텐츠로
보안서약서에 의거하여
강의 내용을 어떠한 사유로도 임의로 복사,
촬영, 녹음, 복제, 보관, 전송하거나
허가 받지 않은 저장매체를
이용한 보관, 제3자에게 누설, 공개,
또는 사용하는 등의 행위를 금합니다.

SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

Day4-1. Sort/Greedy

Sort

Sort는 정렬이다.

Sort에서 중요한 것은 기준

- 기준이 무엇인가? 가 가장 중요하다

Sort를 사용하는 이유

- 다른 알고리즘을 적용하기에 수월하다
- 탐색의 경우, 정렬을 해야 더 빠르게 풀이가 된다 던지

정렬의 종류는 다양하다

- Burble sort
- Merge sort
- Selection sort
- Insertion sort
- Quick Sort 등등

정렬마다 성능의 차이는 있다.

우리 강의에서는 직접 구현하지 않는다.

- C++ STL 에서 제공하는 sort 는 내부적으로 IntroSort 가 돌고, 속도를 보장해줌 (NlogN)

sort 사용하기 1

<algorithm> 헤더 필요

sort(시작주소, 끝주소);

배열 정렬하기

- **배열의 이름은 주소이다.**

```
#include<algorithm>
using namespace std;

int main() {
    int arr[10] = { 12,43,3,2,5,2,3,1,5,6 };
    //sort(배열시작주소, 끝주소);
    sort(arr, arr + 10);
```

https://gist.github.com/hoconoco/b87f1f6430e4c35facff172f9af534dc

vector 정렬하기

- .begin() : vector 시작 주소
- .end() : vector 끝 주소

```
vector<int> a = { 1,45,2,2,5,4,4,67,6,4,5 };


//.begin() : vector 시작 주소
//.end() : vector 끝 주소
sort(a.begin(), a.end());
```

https://gist.github.com/hoconoco/65abb8487f8526b25df5afd992dd659b

sort에서는 cmp 옵션이 있다.

sort() 에는 정렬 옵션을 바꿀 수 있는 cmp 옵션이 있다.

- cmp 함수를 재정의해서 입맛에 맞게 바꿀 수 있다.
- default 로는 오름차순 정렬(less) 이지만, 내림차순을 사용하거나, 기준을 임의로 정할 수 있다
- _Pred : 조건자로 정렬 기준을 사용자 정의할 때 사용한다.

```
_CONSTEXPR20 void sort(const _RanIt _First, const _RanIt _Last, _Pr _Pred) { // order [_First, _Last)
    _STD _Adl_verify_range(_First, _Last);
    const auto _UFirst = _STD _Get_unwrapped(_First);
    const auto _ULast  = _STD _Get_unwrapped(_Last);
    _STD _Sort_unchecked(_UFirst, _ULast, _ULast - _UFirst, _STD _Pass_fn(_Pred));
}

_EXPORT_STD template <class _RanIt>
_CONSTEXPR20 void sort(const _RanIt _First, const _RanIt _Last) { // order [_First, _Last)
    _STD sort(_First, _Last, less<>{});
}
```

[참고] less 의 구조

default 옵션인 less를 확인하면, _Left < _Right 가 기본이다.

- **내림차순으로 변경하고자 한다면, > 로 바꾸면 된다라는 뜻!**

```
struct less {
    using _FIRST_ARGUMENT_TYPE_NAME _CXX17_DEPRECATE_ADAPTOR_TYPEDEFS  = _Ty;
    using _SECOND_ARGUMENT_TYPE_NAME _CXX17_DEPRECATE_ADAPTOR_TYPEDEFS = _Ty;
    using _RESULT_TYPE_NAME _CXX17_DEPRECATE_ADAPTOR_TYPEDEFS          = bool;

    _NODISCARD constexpr bool operator()(const _Ty& _Left, const _Ty& _Right) const
        noexcept(noexcept(_STD _Fake_copy_init<bool>(_Left < _Right))) /* strengthened */ {
        return _Left < _Right;
    }
};

template <>
struct less<void> {
    template <class _Ty1, class _Ty2>
    _NODISCARD constexpr auto operator()(_Ty1&& _Left, _Ty2&& _Right) const
        noexcept(noexcept(static_cast<_Ty1&&>(_Left) < static_cast<_Ty2&&>(_Right))) // strengthened
        -> decltype(static_cast<_Ty1&&>(_Left) < static_cast<_Ty2&&>(_Right)) {
        return static_cast<_Ty1&&>(_Left) < static_cast<_Ty2&&>(_Right);
    }
```

내림차순을 만드는 방법

내림차순을 구현하는 방법은 두 가지나 있다!

1. cmp 옵션 함수를 재정의하는 방법
2. < operator 를 변경하는 방법
   - **구조체를 다루면서 설명**

두 가지 모두 다뤄보자.

내림차순 만들기 - cmp 재정의

cmp()를 재정의 해서 사용한다.

- left > right 로 return 한다.
- 기본 코드 구성을 살펴보고 재해석

```
//내림차순으로 재정의
bool cmp(int left, int right) {
    return left > right;
}

int main() {

    sort(a.begin(), a.end(), cmp);
```

https://gist.github.com/hoconoco/205e1e8ff5dfce22ea756793eb5492f6

내림차순 이외에 더 복잡한 조건도 가능하다

cmp() 의 우선순위를 어떻게 작성하는지에 따라
복잡한 조건도 처리 가능하다

오른쪽 코드를 이해해보자

```
//오름차순 처리하기
bool cmpUP(int left, int right) {
    if (left < right) return true;
    if (left > right) return false;
    return false;
}

//내림차순 처리하기
bool cmpDOWN(int left, int right) {
    if (left > right) return true;
    if (left < right) return false;
    return false;
}
```

https://gist.github.com/hoconoco/025f164cea4672bc167256ce59d8333f

우선순위 추가하기

짝수를 홀수보다 앞에 있고,
그 다음 오름차순 정렬하기

- **우선 처리할 조건을 잘 풀어서 작성한다**

```
bool cmp(int left, int right) {
    //1. 짝수 우선
    if (left % 2 == 0 && right % 2 == 1) return true;
    if (left % 2 == 1 && right % 2 == 0) return false;

    //2. 숫자 크기 비교
    if (left < right) return true;
    if (left > right) return false;
    return false;
}
```

https://gist.github.com/hoconoco/03d21a6a40287f42517d5ec04a3feaec

구조체 정렬하기

여러 타입을 보관할 수 있는 구조체의 요소를 비교해서 정렬해야 응용 문제를 풀 수 있다.

다만, 구조체는 직접 비교할 수 있는 정렬 기준이 없다.

- **방법 1. 옵션 함수를 직접 구현**
- **방법 2. < 연산자를 직접 오버로딩**

```
struct ABC t[4] = {
        {1,3},
        {4,45},
        {2,3},
        {65,3}
};

sort(t, t[4]);
```

구조체 정렬하기 - cmp 사용하기

cmp()를 직접 만든다.

- 똑같다.

```
180 90
170 60
170 80
160 50
```

```
bool cmp(ABC left, ABC right) {
    //1. height 가 큰 것 우선
    if (left.height > right.height) return true;
    if (left.height < right.height) return false;

    //2. x가 작은 것 우선
    if (left.weight < right.weight) return true;
    if (left.weight > right.weight) return false;
    return false;
}
```

https://gist.github.com/hoconoco/7333f373cf2dba9f18f259e0e9ac0a5f

구조체 정렬하기 - 연산자 오버로딩

내부에 작성할 값은 똑같다.

- 연산자 오버로딩을 할 경우에는 구조체 내부에 정의해도 괜찮다.

```
bool operator<(ABC left, ABC right) {
    //1. height 가 큰 것 우선
    if (left.height > right.height) return true;
    if (left.height < right.height) return false;

    //2. x가 작은 것 우선
    if (left.weight < right.weight) return true;
    if (left.weight > right.weight) return false;
    return false;
}
```

https://gist.github.com/hoconoco/5027d38baf155bc99c0f1ab4c29e3149

[참고] 구조체 내부에 연산자 오버로딩

구조체의 경우 구조체를 정의하면서
내부에 연산자 오버로딩을 작성할 수 있다

```
struct ABC {
    int height;
    int weight;
    bool operator<(ABC right) {
        //1. height 가 큰 것 우선
        if (height > right.height) return true;
        if (height < right.height) return false;

        //2. x가 작은 것 우선
        if (weight < right.weight) return true;
        if (weight > right.weight) return false;
        return false;
    }
};
```

https://gist.github.com/hoconoco/d74b15dc8b94e757b6bdc177c1828573

SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

Greedy

탐욕 알고리즘

- 최적해를 구하는 문제를 해결하기 위해 사용된다.
- 매 단계, 가장 최적이라고 생각되는 선택을 반복적으로 수행한다.

조건

- 현재 단계에서의 최적의 선택이 전체 최적해를 보장해야 한다.

대표 유형

- 거스름돈
- 서로 겹치지 않는 활동들의 최대 개수
- 최단 경로 문제

Greedy 알고리즘 구별하기

1. Greedy Choice Property

- 선택의 기준이 변경되면 안된다.

2. Optimal Substructure

- 부분 최적해 → 앞에서 계산된 결과를 그대로 가져가서 쓸 수 있는가?

3. 수학적으로 검증이 가능하다.

- 검증할 수 있어야 한다.

4. 구현한다.

대표 유형 살펴보기 1 - 거스름돈

거스름돈 문제

장사를 하고 있고, 아래와 같이 거슬러줄 수 있는 동전이 있다.

- 500원
- 100원
- 50원
- 10원

손님이 물건을 샀을 때 최소의 동전 개수로 거슬러주는 방법을 구하자

문제 분석

1. Greedy Choice Property (선택 기준이 변경되지 않는다)

가장 큰 단위의 동전을 선택하더라도 이 후 남은 금액에 대해 최적의 해를 구하는 데 영향을 주지 않는다.

500원 → 100원 → 50원 → 10원 순으로 계속 그때 그때 가장 큰 금액 선택

2. Optimal Substructure (부분 최적해)

1,750원을 거슬러주는 문제는 1,500원, 250원 으로 문제를 나눠서 1번의 기준으로 풀이하더라도 1,750원을 거슬러주는 문제에 기여한다.

수학적 검증하기

1,750원을 최소의 동전 개수로 거슬러줘야 한다면?

- 500원 3개 = 1,500원
- 100원 2개 = 200원
- 50원 1개 = 50원
- 10원 0개

총 6개의 동전이 필요하다

단순 사칙연산으로도 풀이가 가능하다

```
//10,50,100,500
int coin[4] = { 500,100,50,10 };

int cnt = 0;
int i = 0;
while (i < 4) {
    cnt += n / coin[i];
    n %= coin[i];
    i++;
}
```

https://gist.github.com/hoconoco/44c2bbed08c9ce99c2b1c46f79554aad

대표 유형 살펴보기 2 - ATM 문제

ATM 문제

ATM기 1대가 있다.
A는 30분, B는 20분, C는 5분을 사용할 때,
A, B, C 가 대기하는 시간의 총 대기시간 중 최소 대기시간을 구하자

대표 유형 살펴보기 2 - ATM 문제

문제 분석

- ATM 사용 순서를 결정하여 **총 대기 시간을 최소화하는 것**입니다.
- 각 사용자가 ATM을 사용하는 데 걸리는 시간이 주어졌을 때, 먼저 처리되는 사람이 남은 사람들의 대기 시간에 영향을 미친다.

선택 기준

사용 시간이 짧은 사람부터 처리하면

전체 대기 시간을 최소화할 수 있습니다.

수학적 검증하기

가장 긴 사용시간부터 선택한다면?

1. A(30) 분 사용, B,C 30분씩 대기) → 총 대기 시간 60분
2. B(20) 분 사용, C 20분 대기 → 총 대기 시간 80분

가장 짧은 사용시간부터 선택한다면?

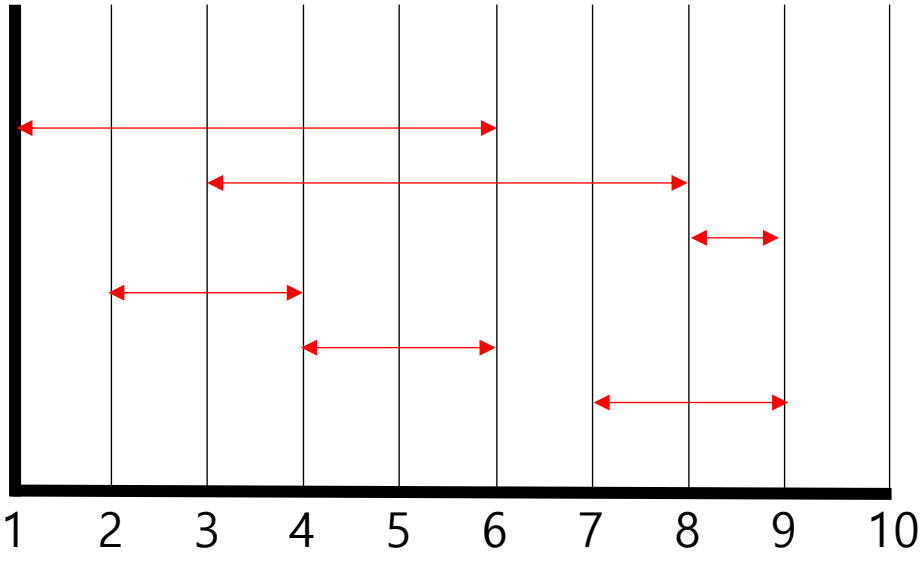
1. C(5)분 사용, A,B 5분씩 대기 → 총 대기시간 10분
2. B(20)분 사용, A 20분 대기 → 총 대기시간 30분

회의실 배정 문제

회의실이 하나인 회사가 있다.

회의 스케줄을 입력 받고 가장 많은 회의가 열리기 위한 총 회의 개수를 구하라

회의 스케줄
시작 끝
1 6
3 8
8 9
2 4
4 6
7 9



대표 유형 살펴보기 3 - 회의실 배정 문제

문제 분석

- 하나의 회의실에서 여러 회의를 진행해야 하는데, **가능한 한 많은 회의를 배정**해야 한다.
- 각 회의는 시작 시간과 종료 시간이 주어지고, **회의 시간이 겹치지 않아야 한**다.
- 목표는 **최대한 많은 회의를 회의실에 배정하는 것**

1. Greedy Choice Property (탐욕 선택 속성)

- 각 단계에서 **종료 시간이 가장 빠른 회의를 먼저 선택하는 전략**이 최적해로 이어진다.
- 종료 시간이 빠른 회의를 선택하면 더 많은 회의를 배정할 수 있는 가능성을 최대화하기 때문

2. Optimal Substructure (최적 부분 구조)

- 전체 문제(모든 회의 배정)의 최적해는 **부분 문제(남은 회의 배정)의 최적해로 구성될** 수 있음
- ex) 종료 시간이 빠른 회의를 선택한 후 남은 회의에서 동일한 전략을 적용하면 최적해를 구할 수 있다.

대표 유형 살펴보기 3 - 회의실 배정 문제

검증하기

1. 제일 시간이 짧은 것 우선
   - 예시 1 : (8,9), (2,4), (4,6) → 3개 O
   - 예시 2 : (5,6) → 1개 X
   - 예시 2와 같은 조건으로 인해 안됨

2. 빨리 종료되는 회의 우선
   - 예시 1 : (2,4), (4,6), (7,9 or 8,9) → 3개 O
   - 예시 2 : (1,6), (6,10) → 2개 O

3. 제일 빨리 시작하는 회의 우선
   - 예시 1 : (1,6), (7,9) → 2개 X
   - 예시 2 : (1,6), (6,10) → 2개 O
   - 예시 1과 같은 조건으로 인해 안됨

회의 스케줄 예시 1
1 6
3 8
8 9
2 4
4 6
7 9

회의 스케줄 예시 2
1 6
6 10
5 6

구조체로 시간 정보를 받아서 정렬한다.

- **정렬 기준 : 가장 빨리 끝나는 회의 시간 기준**

```
bool cmp(struct Schedule left, struct Schedule right) {
    if (left.end < right.end) return true;
    if (left.end > right.end) return false;
    return false;
}
```

https://gist.github.com/hoconoco/6ea22058e64f1b17734e3769b968e8e9

Greedy 알고리즘 구별하기

1. Greedy Choice Property

- 선택의 기준이 변경되면 안된다.

2. Optimal Substructure

- 부분 최적해 → 앞에서 계산된 결과를 그대로 가져가서 쓸 수 있는가?

3. 수학적으로 검증이 가능하다.

- 검증할 수 있어야 한다.

4. 구현한다.

매우 중요하다.
앞으로 학습할 완전탐색과 Greedy를 구분하는 것이
가장 빠르게 문제를 해결하는 방법이 될 수 있다

SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

내일 방송에서 만나요!

삼성 청년 SW 아카데미