TS기지국설치

Hancom Education Co. Ltd. JooHyun – Lee (comkiwer)

Problem

문제 설명

기지국을 설치할 수 있는 건물의 위치가 주어진다.
건물은 모두 일렬로 존재하며, 추가되거나 삭제될 수 있다.

[Fig. 1]은 4개의 건물이 주어진 예이다.
ID가 700인 건물이 좌표 2에 위치하고, ID가 500인 건물은 좌표 7에 위치한다.

Building	700						500						200								300				
Location	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

[Fig. 1]

기지국 사이의 간섭을 최소화하기 위해서,
가장 인접한 기지국 사이의 거리를 최대로 하는 위치에 M개의 기지국을 모두 설치하려고 한다.

문제 설명

예를 들어, [Fig. 1]에 3개의 기지국을 설치하는 경우를 살펴 보자.

건물 700, 건물 500, 건물 300에 기지국을 설치할 경우에는
가장 인접한 기지국이 건물 700과 건물 500에 설치한 기지국이 되고, 거리는 5이다.

하지만 [Fig. 2]와 같이 설치하면
가장 인접한 기지국 사이의 거리가 9(건물 200과 건물 300에 설치된 기지국 사이의 거리)로 최대가 된다.

Building		700						500					200								300			
Location	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

[Fig. 2]

※ 아래 API 설명을 참조하여 각 함수를 구현하라.

문제 설명

void init(int N, int mId[], int mLocation[])

각 테스트 케이스의 처음에 호출된다.

N개의 건물 ID, 건물 위치가 주어진다.

건물 ID가 서로 같은 경우는 없다.
건물 위치가 서로 같은 경우도 없다.

Parameters

N: 건물 개수 (2 ≤ N ≤ 100)

(0 ≤ i < N)인 모든 i에 대해,

mId[i]: 건물 i의 ID (1 ≤ mId[i] ≤ 1,000,000,000)

mLocation[i]: 건물 i의 위치 (1 ≤ mLocation[i] ≤ 1,000,000,000)

문제 설명

int add(int mId, int mLocation)

ID가 mId이고, 건물 위치가 mLocation인 건물을 추가한다.
만약에 이미 존재하는 ID라면, 건물을 추가하지 않고 위치를 변경한다.
mLocation이 다른 건물의 위치 값과 동일하게 주어지는 경우는 없다.

Parameters

mId: 건물 ID (1 ≤ mId ≤ 1,000,000,000)

mLocation: 건물의 위치 (1 ≤ mLocation ≤ 1,000,000,000)

Returns

추가나 위치 변경 이후에, 건물의 총 개수를 반환한다.

문제 설명

int remove(int mStart, int mEnd)

위치가 mStart이상 mEnd이하인 모든 건물을 삭제한다.

Parameters

mStart: 삭제할 범위의 시작 위치 (1 ≤ mStart ≤ 1,000,000,000)

mEnd: 삭제할 범위의 끝 위치 (mStart ≤ mEnd ≤ 1,000,000,000)

Returns

삭제 이후에, 남아 있는 건물의 총 개수를 반환한다.

문제 설명

int install(int M)

남아 있는 건물에 가장 인접한 기지국 사이의 거리를 최대로 M개의 기지국을 모두 설치할 경우,
가장 인접한 기지국 사이의 거리를 반환한다.
건물의 개수보다 M 값이 크게 주어지는 경우는 없다.

Parameters

M: 기지국의 개수 (2 ≤ M ≤ 2,000)

Returns

가장 인접한 기지국 사이의 최대 거리를 반환한다.

문제 설명

[제약사항]

1. 각 테스트 케이스 시작 시 init() 함수가 호출된다.
2. 각 테스트 케이스에서 add() 함수의 호출 횟수는 24,000 이하이다.
3. 각 테스트 케이스에서 remove() 함수의 호출 횟수는 3,000 이하이다.
4. 각 테스트 케이스에서 모든 함수의 호출 횟수 총합은 30,000 이하이다.

문제 설명

[문제 요약]

1차원 좌표(1~10억)에 건물id(1~10억)가 추가 / 삭제된다.

건물들에 **가장 인접한 기지국사이의 거리가 최대가 되도록** **M개의 기지국을 설치할 때**
가장 인접한 기지국사이의 거리를 구하시오. **(한 위치에 2개 이상의 건물이 존재하지 않는다.)**

void init(N, id[], loc[]) : N(100)개의 id와 위치가 주어진다.

int add(id, loc) : id 건물을 loc위치로(추가, 변경). [건물 수 반환] , *24,000 호출*

int remove(sLoc, eLoc) : 위치가 sLoc ~ eLoc인 건물을 모두 삭제하라. [건물 수 반환], *3,000 호출*

int install(M) : M개의 기지국을 조건에 맞게 세운다. [거리 반환)

모든 함수 30,000 호출

Building		700					500						200								300			
Location	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

[Fig. 2]

Problem analysis

문제 분석 : 예제

Order	Function	return
1	init (4, {200, 700, 500, 300}, {13, 2, 7, 22})	

(순서 1) 초기에 4개의 건물 정보가 주어진다. 각각의 ID와 위치는 [Fig. 1]과 같다.

Building	700				500				200									300					
Location	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

[Fig. 1]

문제 분석 : 예제

Order	Function	return
2	install (3)	9

(순서 2) 가장 인접한 기지국 사이의 거리를 최대로 3개의 기지국을 설치하는 방법은 [Fig. 2]와 같다.
가장 인접한 기지국 사이의 최대 거리로 9를 반환한다.

Building		700				500							200									300			
Location	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

[Fig. 2]

문제 분석 : 예제

Order	Function	return
3	add (600, 25)	5

(순서 3) ID가 600이고 위치가 25인 건물이 [Fig. 3]과 같이 추가된다.

건물의 총 개수로 5를 반환한다.

Building	700						500						200								300				600
Location	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

[Fig. 3]

문제 분석 : 예제

Order	Function	return
4	install (3)	11

(순서 4) 가장 인접한 기지국 사이의 거리를 최대로 3개의 기지국을 설치하는 방법은 [Fig. 4]와 같다.

가장 인접한 기지국 사이의 최대 거리로 11을 반환한다.

Building		700				500						200						300					600
Location	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

[Fig. 4]

문제 분석 : 예제

Order	Function	return
5	add (200, 18)	5

(순서 5) ID가 200인 건물이 이미 있기 때문에 위치만 변경한다. 건물의 총 개수로 5를 반환한다.

[Fig. 5]는 함수 호출의 결과를 나타낸 그림이다.

Building	700						500											200				300			600
Location	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

[Fig. 5]

문제 분석 : 예제

Order	Function	return
6	install (4)	5

(순서 6) 가장 인접한 기지국 사이의 거리를 최대로 4개의 기지국을 설치하는 방법은 [Fig. 6]과 같다.

가장 인접한 기지국 사이의 최대 거리로 5를 반환한다.

Building		**700**					**500**											**200**							**600**
Location	1	**2**	3	4	5	6	**7**	8	9	10	11	12	13	14	15	16	17	**18**	19	20	21	22	23	24	**25**

[Fig. 6]

문제 분석 : 예제

Order	Function	return
7	remove (1, 7)	3

(순서 7) 위치가 1이상 7이하인 모든 건물을 삭제한다. 삭제하고 남은 건물의 총 개수로 3을 반환한다.

[Fig. 7]은 함수 호출의 결과를 나타낸 그림이다.

Building	700						500							200					300				600
Location	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

[Fig. 7]

문제 분석 : 예제

Order	Function	return
8	install (2)	7

(순서 8) 가장 인접한 기지국 사이의 거리를 최대로 2개의 기지국을 설치하는 방법은 [Fig. 8]과 같다.

가장 인접한 기지국 사이의 최대 거리로 7을 반환한다.

Building																	**200**							**600**
Location	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	**18**	19	20	21	22	23	24	**25**

[Fig. 8]

문제 분석 : 제약 조건 및 API

- 1차원 좌표(1~10억)에 건물id(1~10억)가 추가 / 삭제된다.
- 건물들에 **가장 인접한 기지국사이의 거리가 최대가 되도록** **M개의 기지국을 설치할 때**
  가장 인접한 기지국사이의 거리를 구하시오. **(한 위치에 2개 이상의 건물이 존재하지 않는다.)**
  1. void init(N, id[], loc[]) : N(100)개의 id와 위치가 주어진다.
  2. int add(id, loc) : id 건물을 loc위치로(추가, 변경). [건물 수 반환] , *24,000 호출*
  3. int remove(sLoc, eLoc) : 위치가 sLoc ~ eLoc인 건물을 모두 삭제하라. [건물 수 반환], *3,000 호출*
  4. int install(M) : M개의 기지국을 조건에 맞게 세운다. [거리 반환)
     모든 함수 30,000 호출

✓ 건물 id에 2 가지 방법으로 접근할 수 있어야 한다.

1. id를 이용하여 접근 : int add(id, loc)
2. 위치를 이용하여 접근 : int remove(sLoc, eLoc), int install(M)

➢ 어떤 자료구조를 써야 할 까?

➢ 핵심 API함수는 어느 것일까?

Solution sketch

해법 연구

- id의 범위가 1~10억 이므로
  id를 이용한 접근을 위하여 unordered_map이 사용될 수 있다. : int add(id, loc)
  unordered_map<first_key: id, second_value: location 또는 set<pii>::iterator>

- 위치를 기준으로 한 삭제와 (: int remove(sLoc, eLoc))
  기지국을 세우기 위하여 (: int install(M))
  위치를 기준으로 정렬된 상태를 유지하는 자료구조로 set이 사용될 수 있다.
  set<pair<first_key:location, second_value: id>

```
const int LM = 30000;
using pii = pair<int, int>;
set<pii> myset;                                  // <fist: location, second: ID>
unordered_map<int, set<pii>::iterator> hmap; // <first: ID, second: location>
int A[LM];                                       // int install(int M) 에서 사용할 임시 배열
```

해법 연구

- ✓ unordered_map과 set을 이용한다면 아래 세 API함수 처리는 비교적 어렵지 않게 처리할 수 있다.
  - ✓ void init(N, id[], loc[])
  - ✓ int add(id, loc)
  - ✓ int remove(sLoc, eLoc)

- 이제 int install(M) API함수는 어떻게 처리할 수 있을까?

  먼저 단순한 방법 생각해 보자.

  구하고자 하는 답은 low(이웃한 건물사이거리 최솟값) ~ high(건물위치 최댓값 - 최솟값) 사이에 있다.

  그렇다면 답 ans를 high부터 시작하여 low까지 가정하며 M개의 기지국을 세울 수 있는지 검사해 볼 수 있다.

  high부터 가정하는 이유는 기지국 사이의 거리를 가능한 최대로 해야 하기 때문이다.

해법 연구

이 방법을 코드로 나타내 보면 다음과 같다.

```
using pii = pair<int, int>;    // pair<first_key: location, second_value: id>
set<pii> myset;
int A[LM];

int install(int M) {
    int low = (int)1e9, high;  // 답으로 가능한 범위 : low ~ high
    int ans, len = 0;          // ans: 답, len: 건물 수
    for (auto&p : myset) A[len++] = p.first; // 임시 배열 A[]에 위치만 복사, 길이(len) 구하기
    for (int i = 1; i < len; ++i)            // 답 범위의 최솟값
        low = min(low, A[i] - A[i - 1]);
    ans = high = A[len - 1] - A[0];          // 답 범위의 최대값

    for( ans = high; ans>=low; --ans){       // 최댓값부터 시작하여 답으로 가능한지 검사.
        int cnt = 1, latest = A[0];          // 첫 위치에 기지국 놓고 시작
        for (int i = 1; i < len; ++i) {
            if (A[i] - latest >= ans)        // 답으로 가정한 ans보다 거리가 늘어나는 경우
                cnt++, latest = A[i];        // 새로운 기지국 설치
        }
        if (cnt == M) break;                 // M개의 기지국이 설치된 경우
    }
    return ans;
}
```

해법 연구

이 코드의 문제점은 무엇일까?

아래 [ ] 표시 부분 때문에 **시간이 너무 많이 걸린다**는 것이다.

이 부분은 최대 약 10억 * len까지 걸릴 수 있다. (예 : low = 1, high = 10억-1, M == len)

```
// ...
int low = (int)1e9, high;  // 답으로 가능한 범위 : low ~ high
int ans, len = 0;          // ans: 답, len: 건물 수
for (auto&p : myset) A[len++] = p.first; // 임시 배열 A[]에 위치만 복사, 길이(len) 구하기
for (int i = 1; i < len; ++i)            // 답 범위의 최솟값
    low = min(low, A[i] - A[i - 1]);
ans = high = A[len - 1] - A[0];          // 답 범위의 최대값

for( ans = high; ans>=low; --ans){       // 최댓값부터 시작하여 답으로 가능한지 검사.
    int cnt = 1, latest = A[0];          // 첫 위치에 기지국 놓고 시작
    for (int i = 1; i < len; ++i) {
        if (A[i] - latest >= ans)        // 답으로 가정한 ans보다 거리가 늘어나는 경우
            cnt++, latest = A[i];        // 새로운 기지국 설치
    }
    if (cnt == M) break;                 // M개의 기지국이 설치된 경우
}
// ...
```

해법 연구

시간을 줄이기 위하여 어떤 방법을 생각할 수 있을까?

다음 두 가지 특성이 있으므로 답을 가정하는 binary search(= parametric search)를 이용할 수 있다.

1. 답으로 가능한 범위(후보군)가 정렬되어 있다.
   low ~ high 중에 답이 반드시 존재한다.
2. 인접한 기지국 사이의 거리를 최대로 하는 최대 거리 ans가 결정되어 구해진 경우 low ~ high구간은
   low ~ ans ~ high 구간으로 나뉘어지며 (물론 low(ans) ~ high, low ~ high(ans) 일 수도 있다.)
   low ~ ans 은 M개의 기지국을 설치할 수 있고, ans+1 이상의 구간은 M개의 기지국을 설치할 수 없다.

아래 예를 보면 ans=5 이하에서는 항상 가능하고 6부터는 항상 불가능함을 알 수 있다.

Order	Function	return
6	install (4)	5

Building		700					500											200			300				600
Location	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

[Fig. 6]

해법 연구

앞서 살펴본 방법을 사용한다면 최대 10억인 후보들을 최대 30번으로 줄일 수 있다.

따라서 이문제의 핵심 API함수는 int install(M) 라고 할 수 있으며
“이진 탐색을 적용했는가?” 라는 것이 주요 허들이 된다.

이제 각 API함수별 할 일을 정리해 보자.

해법 연구

void init(int N, int mId[], int mLocation[])

- hmap과 myset을 초기화 한다.
- N개의 초기 자료를 입력한다.

```
hmap.clear(), myset.clear();                // 초기화
for (int i = 0; i < N; ++i)                 // N개의 초기 자료 설정
    hmap[mId[i]] = myset.insert({ mLocation[i], mId[i] }).first;
```

해법 연구

int add(int mId, int mLocation)

- mId를 hmap에서 찾아 본다.
- 있다면 삭제한다.
- 현재 주어진 mId 데이터를 입력한다.
- 건물 수를 반환한다.

```
auto it = hmap.find(mId);                    // mId 데이터 찾기
if (it != hmap.end())
    myset.erase(it->second);                 // 존재한다면 set에서 삭제하기

// mId 자료를 set과 hmap에 추가하기
hmap[mId] = myset.insert({ mLocation, mId }).first;
return (int)hmap.size();
```

해법 연구

int remove(int mStart, int mEnd)

- mStart이상중에 첫 대상 찾고 mEnd이하까지 자료를 hmap과 myset으로부터 삭제한다.
- 남아 있는 건물 수를 반환한다.

```
auto it = myset.lower_bound({ mStart, 0 });// mStart이상중에 첫 대상 찾기
while (it != myset.end()) {                // mEnd 이하까지 삭제하기
    if (it->first > mEnd) break;
    hmap.erase(it->second);                // hmap에서도 삭제
    it = myset.erase(it);                  // 삭제된 데이터 다음 위치로 업데이트
}
return (int)hmap.size();                   // 건물 수 반환
```

해법 연구

int install(int M)

- 답으로 가능한 범위의 최솟값 low와 최댓값 high를 구한다.
- 건물 위치를 임시 배열 A[]에 담는다.
- low와 high를 이용하여 이진 탐색을 통하여 답 ans를 구한다.

```
for (auto&p : myset)                    // 위치값만 임시배열 A[]에 복사 : set이므로 정렬된 상태로 복사됨.
    A[++len] = p.first;
for (i = 2; i <= len; ++i)              // 답으로 가능한 범위의 최소값 구하기
    low = min(low, A[i] - A[i - 1]);
ans = high = A[len] - A[1];             // 답으로 가능한 범위의 최대값 구하기

while (low <= high) {                   // 이진 탐색
    mid = (low + high) / 2;
    int cnt = 1, latest = A[1];         // 첫 위치에 기지국 놓고 시작
    for (i = 2; i <= len; ++i) {
        if (A[i] - latest >= mid)       // 답으로 가정한 ans보다 거리가 늘어나는 경우
            cnt++, latest = A[i];       // 새로운 기지국 설치
    }
    if (cnt >= M)                       // 답으로 가능한 경우 ans 업데이트: 더 큰 범위로 조정
        ans = mid, low = mid + 1;
    else high = mid - 1;                // 답으로 불가능한 경우 : 더 작은 범위로 조정
}
```

해법 연구

[Summary]

- 큰 정수를 배열의 index처럼 사용하는
  index 기반 프로그래밍을 위하여 unordered_map 을 사용할 수 있다.

- 정렬된 상태를 유지하는 것이 필요한 상황에서 set 컨테이너를 사용할 수 있다.

- 최적화 문제(최댓값, 최솟값)문제를
  문제 분석을 통해
  결정 문제(true, false)문제로 바꾸어 푸는 것이 가능할 때,
  답을 가정하는 **이진 탐색**(parametric search)을 사용할 수 있다.

Code example

Code example

```
#include <set>
#include <vector>
#include <unordered_map>
#include <algorithm>
using namespace std;

const int LM = 30000;
using pii = pair<int, int>;
set<pii> myset;                                 // <fist: location, second: ID>
unordered_map<int, set<pii>::iterator> hmap; // <first: ID, second: location>
int A[LM];                                      // int install(int M) 에서 사용할 임시 배열

void init(int N, int mId[], int mLocation[]) {
    hmap.clear(), myset.clear();                // 초기화
    for (int i = 0; i < N; ++i)                 // N개의 초기 자료 설정
        hmap[mId[i]] = myset.insert({ mLocation[i], mId[i] }).first;
}
```

Code example

```
int add(int mId, int mLocation) {
    auto it = hmap.find(mId);                   // mId 데이터 찾기
    if (it != hmap.end())
        myset.erase(it->second);                // 존재한다면 set에서 삭제하기

    // mId 자료를 set과 hmap에 추가하기
    hmap[mId] = myset.insert({ mLocation, mId }).first;
    return (int)hmap.size();                    // 건물 수 반환
}

int remove(int mStart, int mEnd) {
    auto it = myset.lower_bound({ mStart, 0 });// mStart이상중에 첫 대상 찾기
    while (it != myset.end()) {                 // mEnd 이하까지 삭제하기
        if (it->first > mEnd) break;
        hmap.erase(it->second);                 // hmap에서도 삭제
        it = myset.erase(it);                   // 삭제된 데이터 다음 위치로 업데이트
    }
    return (int)hmap.size();                    // 건물 수 반환
}
```

Code example

```
int install(int M) {
    int low = (int)1e9, mid, high;
    int i, ans, len = 0;

    for (auto&p : myset)                    // 위치값만 임시배열 A[]에 복사 : set이므로 정렬된 상태로 복사됨.
        A[++len] = p.first;
    for (i = 2; i <= len; ++i)              // 답으로 가능한 범위의 최소값 구하기
        low = min(low, A[i] - A[i - 1]);
    ans = high = A[len] - A[1];             // 답으로 가능한 범위의 최대값 구하기

    while (low <= high) {                   // 이진 탐색
        mid = (low + high) / 2;
        int cnt = 1, latest = A[1];         // 첫 위치에 기지국 놓고 시작
        for (i = 2; i <= len; ++i) {
            if (A[i] - latest >= mid)       // 답으로 가정한 ans보다 거리가 늘어나는 경우
                cnt++, latest = A[i];       // 새로운 기지국 설치
        }
        if (cnt >= M)                       // 답으로 가능한 경우 ans 업데이트: 더 큰 범위로 조정
            ans = mid, low = mid + 1;
        else high = mid - 1;                // 답으로 불가능한 경우 : 더 작은 범위로 조정
    }
    return ans;
}
```

Thank you