JooHyun – Lee (comkiwer)

TS사물함배정

Hancom Education Co. Ltd.

Problem

Problem

N개의 사물함을 배정하는 시스템이 있다.

사물함은 동일 간격으로 일렬로 존재하며, 1번부터 N번까지 차례로 번호가 주어진다.

사용자가 입실하는 경우, 아래 규칙에 따라 사물함을 배정한다.

- 모든 사물함이 비어 있는 경우, 1번 사물함을 배정한다.
- 그렇지 않은 경우, 연속된 빈 사물함이 가장 큰 구간을 먼저 선택한다.

Problem

가장 큰 구간이 2개 이상일 경우, 사물함 번호가 작은 구간을 선택한다.
선택된 구간 내에서, 사용 중인 사물함과 거리가 가장 먼 곳의 사물함을 배정한다.
조건에 해당하는 사물함이 2개 이상일 경우, 번호가 작은 사물함을 배정한다.
사용자가 퇴실하는 경우, 해당 사용자에게 배정되었던 사물함을 해제한다.

위와 같이 동작하는 사물함 배정 시스템을 구현하여 보자.

※ 아래 API 설명을 참조하여 각 함수를 구현하라.

Problem

void init(int N)

각 테스트 케이스의 처음에 호출된다.

N개의 사물함이 주어지며, 모든 사물함은 빈 상태이다.

Parameters

N: 사물함의 개수 (8 ≤ N ≤ 100,000,000)

Problem

int arrive(int mId)

mId 사용자가 입실한다.

mId가 사물함을 이용 중인 사용자 ID로 주어지는 경우는 없다.

빈 사물함이 없는 상황에서, 이 함수가 호출되는 경우는 없다.

배정 규칙에 따라 사물함을 배정한다.

배정된 사물함의 번호를 반환한다.

Parameters

mId: 사용자 ID (1 ≤ mId ≤ 1,000,000,000)

Returns

배정된 사물함의 번호를 반환한다.

Problem

int leave(int mId)

mId 사용자가 퇴실한다.

mId는 사물함을 이용 중인 사용자 ID로만 주어진다.

mId 사용자에게 배정되었던 사물함을 해제한다. 즉, 해당 사물함은 빈 사물함이 된다.

해제 후에 빈 사물함의 개수를 반환한다.

Parameters

mId: 사용자 ID (1 ≤ mId ≤ 1,000,000,000)

Returns

빈 사물함의 개수를 반환한다.

Problem

[제약사항]

1. **각 테스트 케이스 시작 시** `init()` **함수가 호출된다.**
2. **각 테스트 케이스에서** `arrive()` **함수의 호출 횟수는** `20,000` **이하이다.**
3. **각 테스트 케이스에서 모든 함수의 호출 횟수 총합은** `35,000` **이하이다.**

Problem analysis

Problem analysis : 예제

(순서 1) 8개의 사물함이 주어진다. 초기에는 사물함이 모두 비어 있다.

(순서 2) 200 사용자가 입실한다. 모든 사물함이 비어 있기 때문에 1번 사물함을 배정한다.

(순서 3) 100 사용자가 입실한다. 연속된 빈 사물함이 가장 큰 구간은 2번 ~ 8번 구간이다.

사용 중인 사물함과 가장 먼 곳은 8번 사물함이므로 8을 반환한다.

[Fig. 1]은 함수 호출의 결과를 나타낸 그림이다.

Order	Function	return	Figure
1	init(8)		
2	arrive(200)	1	
3	arrive(100)	8	Fig. 1

Locker	1	2	3	4	5	6	7	8
User	200							100

[Fig. 1]

Problem analysis : 예제

(순서 4) 700 사용자가 입실한다. 연속된 빈 사물함이 가장 큰 구간은 2번 ~ 7번 구간이다.
사용 중인 사물함과 가장 먼 곳은 4번, 5번 사물함이다.
이 중에서 번호가 작은 4번 사물함을 배정하고 4를 반환한다.

[Fig. 2]는 함수 호출의 결과를 나타낸 그림이다.

Order	Function	return	Figure
4	arrive(700)	4	Fig. 2

Locker	1	2	3	4	5	6	7	8
User	200			700				100

[Fig. 2]

Problem analysis : 예제

(순서 5) 600 사용자가 입실한다.
연속된 빈 사물함이 가장 큰 구간은 5번 ~ 7번 구간이다.
사용 중인 사물함과 가장 먼 곳은 6번 사물함이므로 6을 반환한다.
[Fig. 3]은 함수 호출의 결과를 나타낸 그림이다.

Order	Function	return	Figure
5	arrive(600)	6	Fig. 3

Locker	1	2	3	4	5	6	7	8
User	200			700		600		100

[Fig. 3]

Problem analysis : 예제

(순서 6) 200 사용자가 퇴실한다.
1번 사물함은 빈 사물함이 된다.
빈 사물함의 총 개수로 5를 반환한다.
[Fig. 4]는 함수 호출의 결과를 나타낸 그림이다.

Order	Function	return	Figure
6	leave(200)	5	Fig. 4

Locker	1	2	3	4	5	6	7	8
User				700		600		100

[Fig. 4]

Problem analysis : 예제

(순서 7) 300 사용자가 입실한다.

연속된 빈 사물함이 가장 큰 구간은 1번 ~ 3번 구간이다.

사용 중인 사물함과 가장 먼 곳은 1번 사물함이므로 1을 반환한다.

[Fig. 5]는 함수 호출의 결과를 나타낸 그림이다.

Order	Function	return	Figure
7	arrive(300)	1	Fig. 5

Locker	1	2	3	4	5	6	7	8
User	300			700		600		100

[Fig. 5]

Problem analysis : 예제

(순서 8) 800 사용자가 입실한다.
연속된 빈 사물함이 가장 큰 구간은 2번 ~ 3번 구간이다.
사용 중인 사물함과 가장 먼 곳은 2번, 3번 사물함이다.
이 중에서 번호가 작은 2번 사물함을 배정하고 2를 반환한다.
[Fig. 6]은 함수 호출의 결과를 나타낸 그림이다.

Order	Function	return	Figure
8	arrive(800)	2	Fig. 6

Locker	1	2	3	4	5	6	7	8
User	300	800		700		600		100

[Fig. 6]

Problem analysis : 예제

(순서 9) 200 사용자가 입실한다.
연속된 빈 사물함이 가장 큰 구간은 3번 ~ 3번 구간, 5번 ~ 5번 구간, 7번 ~ 7번 구간이다.
이 중에서 사물함 번호가 작은 3번 ~ 3번 구간을 선택한다.
사용 중인 사물함과 가장 먼 곳은 3번 사물함이므로 3을 반환한다.
[Fig. 7]은 함수 호출의 결과를 나타낸 그림이다.

Order	Function	return	Figure
9	arrive(200)	3	Fig. 7

Locker	1	2	3	4	5	6	7	8
User	300	800	200	700		600		100

[Fig. 7]

Problem analysis : 예제

(순서 10) 600 사용자가 퇴실한다. 6번 사물함은 빈 사물함이 된다.
빈 사물함의 총 개수로 3을 반환한다.

(순서 11) 100 사용자가 퇴실한다. 8번 사물함은 빈 사물함이 된다.
빈 사물함의 총 개수로 4를 반환한다.

[Fig. 8]은 함수 호출의 결과를 나타낸 그림이다.

Order	Function	return	Figure
10	leave(600)	3	
11	leave(100)	4	Fig. 8

Locker	1	2	3	4	5	6	7	8
User	300	800	200	700				

[Fig. 8]

Problem analysis : 예제

(순서 12) 400 사용자가 입실한다.
연속된 빈 사물함이 가장 큰 구간은 5번 ~ 8번 구간이다.
사용 중인 사물함과 가장 먼 곳은 8번 사물함이므로 8을 반환한다.
[Fig. 9]는 함수 호출의 결과를 나타낸 그림이다.

Order	Function	return	Figure
12	arrive(400)	8	Fig. 9

Locker	1	2	3	4	5	6	7	8
User	300	800	200	700				400

[Fig. 9]

Problem analysis : 제약조건 및 API

- 문제는 단순하다.
  N(8 ~ 100,000,000) 개의 사물함이 주어질 때,
  주어진 규칙에 맞게 사물함을 배정하고 회수하는 것이다.

- 사물함 배정의 원칙을 살펴보자.
  1. 공간(연속한 빈 사물함의 개수)의 크기가 큰 것.
  2. 공간의 크기가 같다면 사물함 번호가 작은 것.
  3. 선택된 공간에 1번 사물함이 포함되어 있다면 1번 사물함에 배정
  4. 선택된 공간에 N번 사물함이 포함되어 있다면 N번 사물함에 배정
  5. 그렇지 않은 경우 (s + e)/ 2번 사물함에 배정한다.
     : s(공간의 첫 사물함 번호), e(공간의 마지막 사물함 번호)

Problem analysis : 제약조건 및 API

- 사용자는 id를 갖는다.
  id의 범위가 1 ~ 1,000,000,000 이므로 renumbering하여 사용할 필요가 있다.
  unordered_map<int, int> 를 이용하여 해결할 수 있다.

- 사물함 개수는 1억이고, 전체 함수 호출 수는 35,000 이다.
  단순한 배정 (arrive()) 과 회수 (leave()) 는 많은 시간을 요구한다.
  사물함 배정(arrive())과 사물함 회수(leave()) 의 효율성이 요구된다.

- 어떻게 할 것인가?

Solution sketch

Solution sketch

- 연속한 빈 사물함을 하나의 노드로 다룰 수 있다.
  다음과 같이 정의하여 arrive()를 위한 자료구조에서 사용될 수 있다.

- set를 이용하여 빈 공간에 대한 우선순위를 실시간으로 관리할 수 있다.

```
const int LM = 35005;
struct Data {
    int s, e, len, flag;
    bool operator<(const Data&t)const {
        // 1순위:공간의 크기, 2순위:공간의 시작번호
        return len != t.len ? len > t.len:s < t.s;
    }
}ids[LM];
set<Data> emptySet;              // 빈공간 : 자리 배정이 안된 구간들
```

Solution sketch

- 그런데 leave()로 tg 사물함이 반납되면 tg과 앞뒤로 연속한 빈공간은 하나의 공간으로 합쳐져야 한다.

- 예를 들어 아래 그림에서 4번 사물함이 반납되면 (2 ~3) 공간과 (4), 그리고(5~7) 공간은 하나로 합쳐져서 (2 ~ 7) 빈 공간이 되어야 한다.

Locker	1	2	3	**4**	5	6	7	8
User	200			**700**				100

Locker	1	2	3	4	5	6	7	**8**
User	200							**100**

Solution sketch

- 이렇게 하기 위해서는 전체 공간을 관리하는 목록이 필요하다.
  전체 공간 목록은 나뉘어진 각 구간이 사용 중이든 아니든
  시작 인덱스의 오름차순으로 관리된다.
  이를 이용하여 앞에서 요구되는 프로세스를 수행할 수 있다.

```
const int LM = 35005;
struct Data {
    int s, e, len, flag;
    bool operator<(const Data&t)const {
        return len != t.len ? len > t.len:s < t.s;
    }
}ids[LM];
struct Comp {
    bool operator()(const Data&a, const Data&b)const {
        return a.s < b.s;
    }
};

set<Data, Comp> allSet;          // 전체 공간을 각 구간의 시작인덱스 순으로 관리
set<Data> emptySet;              // 빈공간 : 자리 배정이 안된 구간들
```

Solution sketch

- 이제 각 함수 별 할 일을 정리해 보자.

void init(int N)

- ✓ 전역변수 및 컨테이너를 초기화 한다.
- ✓ 초기 전체 공간 컨테이너와 빈 공간 컨테이너를 세팅한다.

```
// 전역변수 및 컨테이너 초기화, N:빈 사물함수, usedCnt:사용중인 사물함 수
::N = N, usedCnt = idCnt = 0; // idCnt: renumbering id 갯수
allSet.clear(), emptySet.clear(), hmap.clear();

// 초기 셋팅: allSet, emptySet
allSet.insert({ 1, N, N, 0 });  // 초기 전체 공간
emptySet.insert({ 1, N, N, 0 });// 초기 빈 공간
```

Solution sketch

int arrive(int mId)

- ✓ 사용중인 사물함수를 1증가시킨다.
- ✓ 가장 좋은 구간 tg를 얻는다.
- ✓ tg을 전체 목록과 빈공간 목록에서 제거한다.
- ✓ 다음에 따라 분기 처리한다.
  1. 구간이 1을 포함하는 경우 1번 사물함을 배정한다.
  2. 구간이 N을 포함하는 경우 N번 사물함을 배정한다.
  3. 그렇지 않은 경우 (tg.s + tg.e) / 2 위치 사물함을 배정한다.
  4. 각 경우에 새롭게 설정된 빈 공간이 발생하는 경우
     전체 목록과 빈 공간 목록에 추가한다.
  5. 배정된 사물함은 전체 목록에도 추가한다.
- ✓ 배정된 사물함 번호를 반환한다.

Solution sketch

int arrive(int mId) : 계속

```
usedCnt++;                      // 사용중인 갯수 증가
int id = getID(mId);
Data tg = *emptySet.begin();    // 조건에 맞는 가장 좋은 자리
emptySet.erase(tg);             // 빈공간 목록에서 삭제
allSet.erase(tg);               // 전체 목록에서 삭제

int s = tg.s, e = tg.e, ret = (s + e) / 2;
if (s == 1) ret = 1;
else if (e == N) ret = N;
tg.s = tg.e = ret, tg.len = tg.flag = 1;

if (s < ret) {                  //ret 의 앞쪽에 빈공간이 있다면
    Data a = { s, ret - 1, ret - s, 0 };  // 빈공간
    emptySet.insert(a);         // 빈공간 목록에 저장
    allSet.insert(a);           // 전체 목록에 저장
}
if (ret < e) {                  //ret 의 뒤쪽에 빈공간이 있다면
    Data b = { ret + 1, e, e - ret, 0 };  // 빈공간
    emptySet.insert(b);         // 빈공간 목록에 저장
    allSet.insert(b);           // 전체 목록에 저장
}
ids[id] = tg;                   // id에 빈자리 할당
allSet.insert(tg);              // 전체목록에 ids[id] 추가
return ret;
```

Solution sketch

int leave(int mId)

- ✓ 사용중인 사물함 수를 1 감소시킨다.
- ✓ 반납된 사물함 번호 tg를 전체 목록에서 삭제한다.
- ✓ tg의 이전 또는 이후에 연속한 빈 공간을 찾고
  있다면 이들과 합쳐 하나의 공간을 만든다.
  이때 빈 공간 목록과 전체 목록에서 찾은 공간들을 삭제한다.
- ✓ tg를 빈 공간 목록과 전체 목록에 추가한다.
- ✓ 빈 공간 수를 반환한다.

Solution sketch

int leave(int mId) : 계속

```
int leave(int mId) {
    usedCnt--;                          // 사용중인 갯수 감소
    int id = getID(mId);                // renumid 정하기
    Data tg = ids[id];
    auto it = allSet.lower_bound(tg); // tg을 전체 목록에서 찾기
    auto nit = next(it, 1);             // tg다음 노드를 전체 목록에서 찾기
    if (it != allSet.begin()) {
        auto pit = prev(it, 1);         // tg이전 노드를 전체 목록에서 찾기
        if (pit->flag == 0) {           // 이전 노드가 빈 공간이라면
            tg.s = pit->s;              // tg과 병합하기: 구간의 시작위치 조정
            tg.len += pit->len;         // 구간의 크기 조정
            emptySet.erase(*pit);       // 빈공간 목록에서 tg이전노드 삭제
            allSet.erase(pit);          // 전체 목록에서 tg이전노드 삭제
        }
    }
    if (nit != allSet.end() && nit->flag == 0) { // tg다음 노드가 빈 공간이라면
        tg.e = nit->e;                  // tg과 병합하기: 구간의 끝위치 조정
        tg.len += nit->len;             // 구간의 크기 조정
        emptySet.erase(*nit);           // 빈공간 목록에서 tg다음노드 삭제
        allSet.erase(nit);              // 전체 목록에서 tg다음노드 삭제
    }
    allSet.erase(it);                   // 전체 목록에서 ids[id] 삭제
    tg.flag = 0;                        // 구간이 정리된 tg은 빈 공간으로 표시
    allSet.insert(tg);                  // 전체 목록에 tg추가
    emptySet.insert(tg);                // 빈공간 목록에 tg추가
    return N - usedCnt;                 // 빈 공간수 반환
}
```

Solution sketch

[Summary]

- 배열의 인덱스로 사용할 수 없는 큰 정수에 새로운 번호를 부여하기 위하여 unordered_map 을 사용할 수 있다.

- 우선순위 자료를 관리하는 경우에 set을 사용할 수 있다.

- 문제의 요구사항을 분석하고 필요한 자료구조와 알고리즘을 구현할 수 있다.

Code example1

: Real Time Update

set,

unordered_map

Code example1

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

const int LM = 35005;
struct Data {
    int s, e, len, flag;
    bool operator<(const Data&t)const {
        return len != t.len ? len > t.len:s < t.s;
    }
}ids[LM];
struct Comp {
    bool operator()(const Data&a, const Data&b)const {
        return a.s < b.s;
    }
};

set<Data, Comp> allSet;         // 전체 공간을 각 구간의 시작인덱스 순으로 관리
set<Data> emptySet;             // 빈공간 : 자리 배정이 안된 구간들
int N, idCnt;
unordered_map<int, int> hmap;   // renumbering id
int usedCnt;
```

Code example1

```
void init(int N) {
    // 전역변수 및 컨테이너 초기화
    ::N = N, usedCnt = idCnt = 0;
    allSet.clear(), emptySet.clear(), hmap.clear();

    // 초기 셋팅: allSet, emptySet
    allSet.insert({ 1, N, N, 0 });  // 초기 전체 공간
    emptySet.insert({ 1, N, N, 0 });// 초기 빈 공간
}

int getID(int mid) {
    auto it = hmap.find(mid);
    if (it == hmap.end())
        it = hmap.insert({ mid, ++idCnt }).first;
    return it->second;
}
```

Code example1

```
int arrive(int mId) {
    usedCnt++;                      // 사용중인 갯수 증가
    int id = getID(mId);
    Data tg = *emptySet.begin();    // 조건에 맞는 가장 좋은 자리
    emptySet.erase(tg);             // 빈공간 목록에서 삭제
    allSet.erase(tg);               // 전체 목록에서 삭제

    int s = tg.s, e = tg.e, ret = (s + e) / 2;
    if (s == 1) ret = 1;
    else if (e == N) ret = N;
    tg.s = tg.e = ret, tg.len = tg.flag = 1;

    if (s < ret) {
        Data a = { s, ret - 1, ret - s, 0 };  // 빈공간
        emptySet.insert(a);         // 빈공간 목록에 저장
        allSet.insert(a);           // 전체 목록에 저장
    }
    if (ret < e) {
        Data b = { ret + 1, e, e - ret, 0 };  // 빈공간
        emptySet.insert(b);         // 빈공간 목록에 저장
        allSet.insert(b);           // 전체 목록에 저장
    }
    ids[id] = tg;                   // id에 빈자리 할당
    allSet.insert(tg);              // 전체목록에 ids[id] 추가
    return ret;
}
```

Code example1

```
int leave(int mId) {
    usedCnt--;                          // 사용중인 갯수 감소
    int id = hmap[mId];
    Data tg = ids[id];
    auto it = allSet.lower_bound(tg); // tg을 전체 목록에서 찾기
    auto nit = next(it, 1);             // tg다음 노드를 전체 목록에서 찾기
    if (it != allSet.begin()) {
        auto pit = prev(it, 1);         // tg이전 노드를 전체 목록에서 찾기
        if (pit->flag == 0) {           // 이전 노드가 빈 공간이라면
            tg.s = pit->s;              // tg과 병합하기: 구간의 시작위치 조정
            tg.len += pit->len;         // 구간의 크기 조정
            emptySet.erase(*pit);       // 빈공간 목록에서 tg이전노드 삭제
            allSet.erase(pit);          // 전체 목록에서 tg이전노드 삭제
        }
    }
    if (nit != allSet.end() && nit->flag == 0) { // tg다음 노드가 빈 공간이라면
        tg.e = nit->e;                  // tg과 병합하기: 구간의 끝위치 조정
        tg.len += nit->len;             // 구간의 크기 조정
        emptySet.erase(*nit);           // 빈공간 목록에서 tg다음노드 삭제
        allSet.erase(nit);              // 전체 목록에서 tg다음노드 삭제
    }
    allSet.erase(it);                   // 전체 목록에서 ids[id] 삭제
    tg.flag = 0;                        // 구간이 정리된 tg은 빈 공간으로 표시
    allSet.insert(tg);                  // 전체 목록에 tg추가
    emptySet.insert(tg);                // 빈공간 목록에 tg추가
    return N - usedCnt;                 // 빈 공간수 반환
}
```

Code example2

: lazy update

set, priority_que

unordered_map

Code example2

```
#include <set>
#include <queue>
#include <unordered_map>
using namespace std;

const int LM = 35005;
struct Data {
    int s, e, len, flag;
    bool operator<(const Data&t)const {
        return len != t.len ? len < t.len : s > t.s;
    }
}ids[LM];
struct Comp {
    bool operator()(const Data&a, const Data&b)const {
        return a.s < b.s;
    }
};

set<Data, Comp> allSet;         // 전체 공간을 각 구간의 시작인덱스 순으로 관리
priority_queue<Data> pq;        // 자리 배정이 안된 구간들
int N, idCnt;
unordered_map<int, int> hmap;   // renumbering id
int usedCnt;
```

Code example2

```
void init(int N) {
    // 전역변수 및 컨테이너 초기화
    ::N = N, usedCnt = idCnt = 0;
    allSet.clear(), pq = {}, hmap.clear();

    // 초기 셋팅: allSet, pq
    allSet.insert({ 1, N, N, 0 }); // 초기 공간
    pq.push({ 1, N, N, 0 });       // 초기 빈 공간
}

int getID(int mid) {
    auto it = hmap.find(mid);
    if (it == hmap.end())
        it = hmap.insert({ mid, ++idCnt }).first;
    return it->second;
}
```

Code example2

```
int arrive(int mId) {
    usedCnt++;          // 사용중인 갯수 증가
    int id = getID(mId);
    Data tg;
    while (!pq.empty()) {
        tg = pq.top();  // 조건에 맞는 가장 좋은 자리
        pq.pop();
        auto it = allSet.find(tg); // 원본 찾기
        if (it == allSet.end()) continue;      // 존재하지 않는 경우
        if (it->e == tg.e && !it->flag) break; // 찾은 경우
    }
    allSet.erase(tg);   // 전체 목록에서 삭제
    int s = tg.s, e = tg.e, ret = (s + e) / 2;
    if (s == 1) ret = 1;
    else if (e == N) ret = N;
    tg.s = tg.e = ret, tg.len = tg.flag = 1;
    if (s < ret) {
        Data a = { s, ret - 1, ret - s, 0 };  // 빈공간
        pq.push(a);         // 빈공간 목록에 저장
        allSet.insert(a);   // 전체 목록에 저장
    }
    if (ret < e) {
        Data b = { ret + 1, e, e - ret, 0 };  // 빈공간
        pq.push(b);         // 빈공간 목록에 저장
        allSet.insert(b);   // 전체 목록에 저장
    }
    ids[id] = tg;           // id에 빈자리 할당
    allSet.insert(tg);      // 전체목록에 ids[id] 추가
    return ret;
}
```

Code example2

```
int leave(int mId) {
    usedCnt--;                          // 사용중인 갯수 감소
    int id = getID(mId);
    Data tg = ids[id];
    auto it = allSet.lower_bound(tg); // tg을 전체 목록에서 찾기
    auto nit = next(it, 1);             // tg다음 노드를 전체 목록에서 찾기
    if (it != allSet.begin()) {
        auto pit = prev(it, 1);         // tg이전 노드를 전체 목록에서 찾기
        if (pit->flag == 0) {           // 이전 노드가 빈 공간이라면
            tg.s = pit->s;              // tg과 병합하기: 구간의 시작위치 조정
            tg.len += pit->len;         // 구간의 크기 조정
            allSet.erase(pit);          // 전체 목록에서 tg이전노드 삭제
        }
    }
    if (nit != allSet.end() && nit->flag == 0) { // tg다음 노드가 빈 공간이라면
        tg.e = nit->e;                  // tg과 병합하기: 구간의 끝위치 조정
        tg.len += nit->len;             // 구간의 크기 조정
        allSet.erase(nit);              // 전체 목록에서 tg다음노드 삭제
    }
    allSet.erase(it);                   // 전체 목록에서 ids[id] 삭제
    tg.flag = 0;                        // 구간이 정리된 tg은 빈 공간으로 표시
    allSet.insert(tg);                  // 전체 목록에 tg추가
    pq.push(tg);                        // 빈공간 목록에 tg추가
    return N - usedCnt;                 // 빈 공간수 반환
}
```

Code example3

: lazy update

custom_DLL

priority_que

unordered_map

Code example3

```
#include <vector>
#include <queue>
#include <unordered_map>
using namespace std;

const int LM = 105005;
struct Node {
    int s, e, len, flag;
    Node*prev, *next;
    void alloc(Node*np, Node*nn) {
        prev = np, next = nn;
        if (np) prev->next = this;
        if (nn) next->prev = this;
    }
    void erase() {
        flag = 1;
        if (prev) prev->next = next;
        if (next) next->prev = prev;
    }
}buf[LM];
```

Code example3

```
struct Comp {
    bool operator()(const Node*a, const Node*b)const {
        return a->len != b->len ? a->len < b->len : a->s > b->s;
    }
};

priority_queue<Node*, vector<Node*>, Comp> pq;
unordered_map<int, Node*> hmap;
int N, bCnt, usedCnt;

void init(int N) {
    ::N = N, bCnt = usedCnt = 0;
    pq = {}, hmap.clear();
    buf[++bCnt] = { 1, N, N, 0 };
    pq.push(buf + bCnt);
}
```

Code example3

```
int arrive(int mId) {
    usedCnt++;
    while (pq.top()->flag)
        pq.pop();
    Node*tg = pq.top();
    pq.pop();

    int s = tg->s, e = tg->e, ret = (s + e) / 2;
    if (s == 1) ret = 1;       // revise result
    else if (e == N) ret = N; // revise result

    tg->s = tg->e = ret, tg->len = tg->flag = 1;
    hmap[mId] = tg;

    if (s < ret) {
        buf[++bCnt] = { s, ret - 1, ret - s, 0 };
        buf[bCnt].alloc(tg->prev, tg);
        pq.push(&buf[bCnt]);
    }
    if (ret < e) {
        buf[++bCnt] = { ret + 1, e, e - ret, 0 };
        buf[bCnt].alloc(tg, tg->next);
        pq.push(&buf[bCnt]);
    }
    return ret;
}
```

Code example3

```
int leave(int mId) {
    usedCnt--;
    Node*tg = hmap[mId], *prev = tg->prev, *next = tg->next;
    tg->flag = 0;
    if (prev && prev->flag == 0) {
        tg->s = prev->s, tg->len += prev->len;
        prev->erase();
    }
    if (next && next->flag == 0) {
        tg->e = next->e, tg->len += next->len;
        next->erase();
    }
    pq.push(tg);

    return N - usedCnt;
}
```

Thank you.