JooHyun – Lee (comkiwer)

TS호텔검색서비스

Hancom Education Co. Ltd.

Problem

Problem

고객이 원하는 호텔의 룸을 빠르게 찾아주는 기능을 구현해 보자.

시스템에 등록된 호텔의 개수 N 은 최대 1,000 이다.

호텔의 ID 값은 1부터 N 사이의 값이며, 중복되지 않는다.

각 호텔은 최대 100 개의 룸을 가지고 있다.

각 룸의 ID 값은 1부터 100,000 사이의 값이며, 중복되지 않도록 주어진다.

(단, 호텔의 ID 값과 룸의 ID 값은 서로 같을 수도 있다.)

Problem

각 룸은 다음과 같은 정보를 가지고 있다.

정보	값의 범위
지역	1 ~ 10
침대 수	2 ~ 10
룸 타입	1 ~ 4
조망 타입	1 ~ 4
초기 가격	10,000 ~ 20,000

Problem

호텔 검색 서비스는, 고객이 입력한 조건에 일치하는 룸 중, 가장 저렴한 룸을 찾아서 알려준다.

고객이 입력하는 정보는 다음과 같다.

정보	값의 범위
체크인 날짜	1 ~ 9,999
체크아웃 날짜	체크인 날짜+1 ~ 10,000
지역	1 ~ 10
침대 수	2 ~ 10
룸 타입	1 ~ 4
조망 타입	1 ~ 4

다음은 구현해야 할 API이다.

Problem

TS호텔검색서비스

void init(int N, int roomCnt[])

이 함수는 각 테스트 케이스에서 맨 처음 한 번 호출된다.

호텔의 개수 N 은 최대 1,000 이다.

roomCnt[] 는 각 호텔이 가지고 있는 룸의 개수를 전달하는 배열이다.

roomCnt[k] 는 호텔 ID k+1 인 호텔이 보유하고 있는 룸의 개수 이다. (0 ≤ k ≤ N-1)

Parameters

N : 호텔의 개수 (1 ≤ N ≤ 1,000)

roomCnt : 각 호텔이 보유하고 있는 룸의 개수. 1 이상 100 이하의 값을 가진다.

Problem

void addRoom(int hotelID, int roomID, int roomInfo[])

호텔에 새로운 룸을 추가 한다.

룸을 추가할 호텔의 ID 값은 hotelID 이다.

추가될 룸의 ID 값은 roomID 이다.

roomInfo 는 추가할 룸에 대한 정보이다.

roomInfo 는 길이가 5인 배열이며, 각 인덱스에는 다음의 값을 포함한다.

[0] : 지역, [1] : 침대 수, [2] : 룸 타입, [3] : 조망 타입, [4] : 가격

각 배열의 값은 본문에 표기한 제약사항을 만족한다.

추가될 룸의 개수는 init() 에서 전달된 룸의 개수를 초과하지 않음이 보장된다.

Parameters

hotelID : 룸을 추가할 호텔의 ID ($1 \le hotelID \le N$)

roomID : 룸 ID ($1 \le roomID \le 100,000$)

roomInfo : 룸의 정보

Problem

TS호텔검색서비스

int findRoom(int filter[])

룸을 검색하고 예약한다. filter 는 검색 조건을 담고 있는 배열이다.

[0] : 체크인 날짜, [1] : 체크아웃 날짜, [2] : 지역, [3] : 침대 수, [4] : 룸 타입, [5] : 조망 타입

각 배열의 값은 본문에 표기한 제약사항을 만족한다.

체크인 ~ 체크아웃 기간에 이미 예약이 되어 있는 룸은 예약할 수 없다.

단, 체크아웃하는 날짜에, 다른 고객이 체크인 하는 것은 가능하다.

검색 조건에 맞는 룸 중, 가장 가격이 싼 룸을 예약한다.

가격이 같을 경우, 그 중 ID 값이 작은 룸을 선택한다.

예약한 룸의 ID 값을 반환한다. 예약 가능한 룸이 없다면 -1을 반환한다.

Parameters

filter : 룸 검색조건

Return

예약한 룸의 ID 값

Problem

int riseCosts(int hotelID)

한 호텔이 보유하고 있는 모든 룸의 가격이 각각 10%씩 오른다.

계산 시 소수점은 버린다.

Parameters

hotelID : 가격이 상승하는 호텔의 ID (1 ≤ hotelID ≤ N)

Return

호텔이 보유하고 있는 모든 룸의 가격의 합

Problem

[제약사항]

1. 호텔의 수 N 은 최대 1,000 이다.
2. init() 함수는 각 테스트케이스의 제일 처음 1회만 호출된다.
3. addRoom() 은 init() 이후에 일괄적으로 호출되며,
   findRoom() 이나 riseCosts() 가 호출 된 이후에는 호출되지 않는다.
4. 각 호텔이 보유한 룸의 개수는 최대 100 이다.
5. findRoom() 의 호출 횟수는 최대 10,000 이다
6. riseCosts() 의 호출 횟수는 최대 500 이다.

Problem analysis

Problem analysis : 예제

호텔의 수 N = 3 이며, 각 호텔이 보유한 룸의 개수는 각각 2, 6, 10 개이다.

Order	Function	Return
1	init(3, {2, 6, 10})	
2	addRoom(1, 1, {2, 2, 2, 1, 14629})	
3	addRoom(1, 2, {2, 3, 1, 2, 10203})	
4	addRoom(2, 3, {2, 3, 1, 1, 15374})	
5	addRoom(2, 4, {1, 2, 2, 1, 10795})	
6	addRoom(2, 5, {2, 2, 1, 2, 17701})	
7	addRoom(2, 6, {1, 3, 2, 2, 13757})	
8	addRoom(2, 7, {1, 3, 1, 2, 13569})	
9	addRoom(2, 8, {1, 3, 2, 2, 17424})	
10	addRoom(3, 9, {1, 3, 2, 1, 18267})	
11	addRoom(3, 10, {2, 3, 1, 2, 11378})	
12	addRoom(3, 11, {1, 3, 1, 1, 11800})	
13	addRoom(3, 12, {2, 3, 2, 2, 16613})	
14	addRoom(3, 13, {1, 3, 2, 1, 13157})	
15	addRoom(3, 14, {2, 3, 1, 2, 10064})	
16	addRoom(3, 15, {1, 3, 2, 1, 12999})	
17	addRoom(3, 16, {1, 3, 1, 2, 13642})	
18	addRoom(3, 17, {2, 2, 2, 2, 14388})	
19	addRoom(3, 18, {2, 2, 2, 2, 16089})	

지역	침대 수	룸 타입	조망 타입	룸 가격	룸 ID
1	2	2	1	10,795	4
	3	1	1	11,800	11
			2	13,569	7
				13,642	16
		2	1	12,999	15
				13,157	13
				18,267	9
			2	13,757	6
				17,424	8
2	2	1	2	17,701	5
		2	1	14,629	1
			2	14,388	17
				16,089	18
	3	1	1	15,374	3
			2	10,064	14
				10,203	2
				11,378	10
		2	2	16,613	12

1번 호텔 방ID
1,2

2번 호텔 방ID
3,4,5,6,7,8

3번 호텔 방ID
9,10,11,12,13,14,15,16,17,18

Problem analysis : 예제

TS호텔검색서비스

Order	Function	Return
20	findRoom({7, 11, 2, 3, 1, 2})	14

지역	침대수	룸타입	조망타입	룸 가격	룸 ID
1	2	2	1	10,795	4
	3	1	1	11,800	11
			2	13,569	7
				13,642	16
		2	1	12,999	15
				13,157	13
				18,267	9
			2	13,757	6
				17,424	8
2	2	1	2	17,701	5
		2	1	14,629	1
			2	14,388	17
				16,089	18
	3	1	1	15,374	3
			2	10,064	14
				10,203	2
				11,378	10
		2	2	16,613	12

{ 7, 11 }

1번 호텔 방ID
1,2

2번 호텔 방ID
3,4,5,6,7,8

3번 호텔 방ID
9,10,11,12,13,14,15,16,17,18

Problem analysis : 예제

Order	Function	Return
21	findRoom({4, 9, 2, 3, 2, 1})	-1

지역	침대수	룸타입	조망타입	룸 가격	룸 ID
1	2	2	1	10,795	4
	3	1	1	11,800	11
			2	13,569	7
				13,642	16
		2	1	12,999	15
				13,157	13
				18,267	9
			2	13,757	6
				17,424	8
2	2	1	2	17,701	5
		2	1	14,629	1
			2	14,388	17
				16,089	18
	3	1	1	15,374	3
			2	10,064	14
				10,203	2
				11,378	10
		2	2	16,613	12

{ 7, 11 }

1번 호텔 방ID
1,2

2번 호텔 방ID
3,4,5,6,7,8

3번 호텔 방ID
9,10,11,12,13,14,15,16,17,18

Problem analysis : 예제

TS호텔검색서비스

Order	Function	Return
22	findRoom({12, 13, 2, 3, 1, 2})	14

지역	침대수	룸타입	조망타입	룸 가격	룸 ID
1	2	2	1	10,795	4
	3	1	1	11,800	11
			2	13,569	7
				13,642	16
		2	1	12,999	15
				13,157	13
				18,267	9
			2	13,757	6
				17,424	8
2	2	1	2	17,701	5
		2	1	14,629	1
			2	14,388	17
				16,089	18
	3	1	1	15,374	3
			2	10,064	14
				10,203	2
				11,378	10
		2	2	16,613	12

(7, 11) (12, 13)

1번 호텔 방ID
1,2

2번 호텔 방ID
3,4,5,6,7,8

3번 호텔 방ID
9,10,11,12,13,14,15,16,17,18

Problem analysis : 예제

TS호텔검색서비스

Order	Function	Return
23	findRoom({21, 25, 1, 2, 2, 2})	-1

지역	침대수	룸타입	조망타입	룸 가격	룸 ID
1	2	2	1	10,795	4
	3	1	1	11,800	11
			2	13,569	7
				13,642	16
		2	1	12,999	15
				13,157	13
				18,267	9
			2	13,757	6
				17,424	8
2	2	1	2	17,701	5
		2	1	14,629	1
			2	14,388	17
				16,089	18
	3	1	1	15,374	3
			2	10,064	14
				10,203	2
				11,378	10
		2	2	16,613	12

[7, 11] [12, 13]

1번 호텔 방ID
1,2

2번 호텔 방ID
3,4,5,6,7,8

3번 호텔 방ID
9,10,11,12,13,14,15,16,17,18

Problem analysis : 예제

Order	Function	Return
24	findRoom({21, 24, 2, 2, 1, 2})	5

지역	침대수	룸타입	조망타입	룸 가격	룸 ID
1	2	2	1	10,795	4
	3	1	1	11,800	11
			2	13,569	7
				13,642	16
		2	1	12,999	15
				13,157	13
				18,267	9
			2	13,757	6
				17,424	8
2	2	1	2	17,701	5
		2	1	14,629	1
			2	14,388	17
				16,089	18
	3	1	1	15,374	3
			2	10,064	14
				10,203	2
				11,378	10
		2	2	16,613	12

[21, 24]

[7, 11] [12, 13]

1번 호텔 방ID
1,2

2번 호텔 방ID
3,4,5,6,7,8

3번 호텔 방ID
9,10,11,12,13,14,15,16,17,18

Problem analysis : 예제

Order	Function	Return
25	findRoom({9, 13, 1, 2, 1, 2})	-1

지역	침대수	룸타입	조망타입	룸 가격	룸 ID	
1	2	2	1	10,795	4	
	3	1	1	11,800	11	
			2	13,569	7	
				13,642	16	
		2	1	12,999	15	
				13,157	13	
				18,267	9	
			2	13,757	6	
				17,424	8	
2	2	1	2	17,701	5	[21, 24]
		2	1	14,629	1	
			2	14,388	17	
				16,089	18	
	3	1	1	15,374	3	
			2	10,064	14	[7, 11] [12, 13]
				10,203	2	
				11,378	10	
		2	2	16,613	12	

1번 호텔 방ID
1,2

2번 호텔 방ID
3,4,5,6,7,8

3번 호텔 방ID
9,10,11,12,13,14,15,16,17,18

Problem analysis : 예제

TS호텔검색서비스

Order	Function	Return
26	riseCosts(2)	97,479

지역	침대수	룸타입	조망타입	룸 가격	룸 ID
1	2	2	1	10,795→11,874	4
	3	1	1	11,800	11
			2	13,569→14,925	7
				13,642	16
		2	1	12,999	15
				13,157	13
				18,267	9
			2	13,757→15,132	6
				17,424→19,166	8
2	2	1	2	17,701→19,471	5
		2	1	14,629	1
			2	14,388	17
				16,089	18
	3	1	1	15,374→16,911	3
			2	10,064	14
				10,203	2
				11,378	10
		2	2	16,613	12

[21, 24]

[7, 11] [12, 13]

1번 호텔 방ID
1,2

2번 호텔 방ID
3,4,5,6,7,8

3번 호텔 방ID
9,10,11,12,13,14,15,16,17,18

Problem analysis : 제약 조건 및 API

- 호텔 최대 1,000개, 호텔 ID 1 ~ 1,000, 호텔 당 룸 수 최대 100 : init()
- 룸ID 1~100,000 순차적으로 주어진다. : addRoom()
- 방개수만큼 addRoom()이 일괄적으로 호출된다.
- 이후 findRoom() 최대 10,000번 riseCosts() 최대 500번 호출된다.
- 호텔 당 룸 수는 최대 100이므로 riseCosts()호출에 대하여
  전 탐색(full search)을 사용할 수 있을 것으로 보인다.

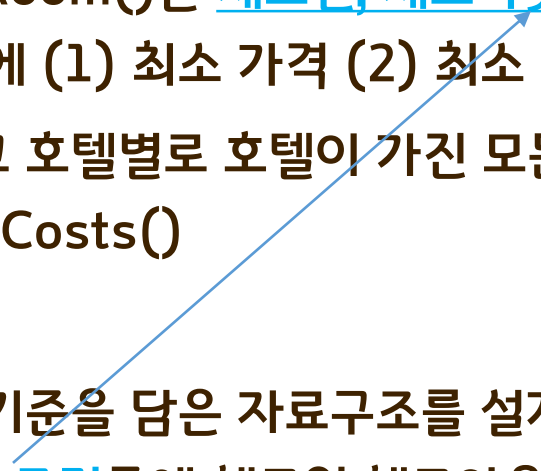
- findRoom()은 체크인, 체크아웃, 지역, 침대 수, 룸 타입, 조망 타입 을 만족하는
  룸 중에 (1) 최소 가격 (2) 최소 ID를 만족하는 ID를 찾아야 한다.

  어떤 자료구조를 이용하여 어떻게 설계할까?

Solution sketch

Solution sketch

- findRoom()은 체크인, 체크아웃, 지역, 침대 수, 룸 타입, 조망 타입 을 만족하는 룸 중에 (1) 최소 가격 (2) 최소 ID를 만족하는 ID를 찾아야 한다.
- 그리고 호텔별로 호텔이 가진 모든 방의 대여료가 10% 증가할 수 있다.
  : riseCosts()



- 모든 기준을 담은 자료구조를 설계하는 것은 쉬운 일이 아니다.
  6개의 조건중에 체크인 체크아웃을 만족하는 범위는 너무 넓어
  딱 떨어지도록 범위를 나누기가 쉽지 않다.

  하지만 지역(1~10), 침대 수(2~10), 룸 타입(1~4), 조망 타입(1~4)은
  [11][11][5][5] 4차원 배열로 분류할 수 있다.

Solution sketch

- 따라서 [지역][침대 수][룸 타입][조망 타입]으로 분류하고 각 분류마다
  (1)가격의 오름차순 (2) 룸ID의 오름차순으로 정렬된 상태를 유지하는 자료를 설계한 후
  각 룸 별로 예약 상황 목록을 관리하도록 설계할 수 있다.
- 이경우 일치하는 분류를 처음부터 순회하며
  체크인, 체크 아웃을 만족하는 룸 ID를 구할 수 있다.

```
using pii = pair<int, int>;
set<pii> ids[11][11][5][5];  // <cost, id>'s ascending
```

Solution sketch

- 각 룸은 어떤 속성을 가져야 할까?
  룸이 가지는 지역, 침대 수, 룸 유형, 조망 유형, 대여료, ID등이 필요할 것이다.
  또한 예약 목록이 필요할 것이다.
- ids[][][][]에서 삭제 시 속도 향상을 위하여 저장위치를 백업해 둘 수 있다.

```
struct Data {
    int region, bedCnt, roomType, viewType, cost, id;


    // ids[region][bedCnt][roomType][viewType]'에 삽입된 위치
    set<pii>::iterator it;
    set<pii> reserveList;  // 예약 목록 <inTime, outTime>
}rooms[RLM];
```

- 이제 각 API별 할 일을 정리해 보자.

Solution sketch

void init(int N, int roomCnt[])

- 호텔의 수와 호텔 별 룸 수는 무시될 수 있다.
- 앞서 살펴본 자료구조를 초기화 한다.

```
void init(int N, int bedCnt[]) {
    // 호텔별 방목록 초기화
    for (int i = 1; i < HLM; ++i) idsByHotel[i].clear();

    // ids[지역][방갯수][방유형][조망유형] 셋 초기화
    for (int i = 1; i < 11; ++i) for (int j = 2; j < 11; ++j) {
        for (int r = 1; r < 5; ++r) for (int c = 1; c < 5; ++c)
            ids[i][j][r][c].clear();
    }
}
```

Solution sketch

void addRoom(int hotelID, int roomID, int roomInfo[])

- **호텔별 룸 번호를 추가한다.**

  ```
  idsByHotel[hotelID].push_back(roomID);
  ```

- **룸 번호 저장, 분류 별 우선순위에 맞게 저장한다.**

  ```
  rooms[roomID].init(roomInfo, roomID);
  ```

```
struct Data {
    int region, bedCnt, roomType, viewType, cost, id;
    set<pii>::iterator it; // ids[region][bedCnt][roomType][viewType]'에 삽입된 위치
    set<pii> reserveList;  // 예약 목록을 <inTime, outTime> 오름차순으로 관리

    void init(int*info, int nid) {
        region = info[0], bedCnt = info[1], roomType = info[2];
        viewType = info[3], cost = info[4], id = nid;
        reserveList.clear();
        // 분류별로 우선순위에 맞게 삽입 : 삽입위치 저장
        it = ids[region][bedCnt][roomType][viewType].insert({ cost, id }).first;
    }
    ...
} rooms[RLM];
```

Solution sketch

int findRoom(int filter[])

- ids[지역][침대 수][방 유형][조망 유형] 목록을 순회하여 우선순위 높은 순으로 예약을 시도해 본다. 예약에 성공한 경우 룸 ID를 실패한 경우 -1을 반환한다.

```
int findRoom(int filter[]) {
    int re = filter[2], be = filter[3], rt = filter[4], vt = filter[5];
    for (auto pa : ids[re][be][rt][vt]) {// 분류별, 우선순위에 따라 일정에 맞는 방 찾기
        if (rooms[pa.second].reserve(filter[0], filter[1]) > 0)
            return pa.second;
    }
    return -1;
}

struct Data {
    ...
     int reserve(int inTime, int outTime) {
        auto it = reserveList.lower_bound({ inTime, 0 });              // 삽입 가능한 위치 찾기
        if (it != reserveList.begin() && inTime < prev(it, 1)->second) // 직전 일정과 겹치는가?
            return -1;
        if (it != reserveList.end() && it->first < outTime)            // 직후 일정과 겹치는가?
            return -1;
        reserveList.insert({ inTime, outTime });                       // 새로운 일정 삽입
        return id;
    }
}rooms[RLM];
```

Solution sketch

int riseCosts(int hotelID)

- hotelID에 속한 모든 룸의 가격을 각각 10% 인상시킨다.

```
int riseCosts(int hotelID) {                    // hotelID 소속의 방 대여료 인상시키기
    int sum = 0;
    for (int id : idsByHotel[hotelID]) sum += rooms[id].updateCost();
    return sum;
}

struct Data {
    ...
     int updateCost() {
        ids[region][bedCnt][roomType][viewType].erase(it); // 1. 먼저 지우고
        cost += cost / 10; // 2. 비용 업데이트 하고
        it = ids[region][bedCnt][roomType][viewType].insert({ cost, id }).first; // 3. 다시 삽입하기
        return cost;
    }
    ...
}rooms[RLM];
```

Solution sketch

TS호텔검색서비스

[Summary]

- 여러 속성을 만족시키는 우선순위 자료구조에 set 배열을 사용할 수 있다.
- 실시간 업데이트가 필요한 자료구조에 set 을 사용할 수 있다.
- set에서 탐색을 위하여 lower_bound를 사용할 수 있다.

- set을 문제 해결에 사용할 수 있는 좋은 문제의 예이다.
- 문제해결에 set을 활용할 수 있도록 확실히 준비해 두자.

Code example

: stl set, vector

Code example

TS호텔검색서비스

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

using pii = pair<int, int>;
const int RLM = 100002;
const int HLM = 1002;

int N;
vector<int> idsByHotel[HLM];
set<pii> ids[11][11][5][5];          // <cost, id>'s asceding
struct Data {
    int region, bedCnt, roomType, viewType, cost, id;
    set<pii>::iterator it;           // ids[region][bedCnt][roomType][viewType]'에 삽입된 위치
    set<pii> reserveList;            // 예약 목록

    void init(int*info, int nid) {
        region = info[0], bedCnt = info[1], roomType = info[2];
        viewType = info[3], cost = info[4], id = nid;
        reserveList.clear();
        // 분류별로 우선순위에 맞게 삽입 : 삽입위치 저장
        it = ids[region][bedCnt][roomType][viewType].insert({ cost, id }).first;
    }
```

Code example


```
    int updateCost() {
        // 1. 먼저 지우고
        ids[region][bedCnt][roomType][viewType].erase(it);
        // 2. 비용 업데이트 하고
        cost += cost / 10;
        // 3. 다시 삽입하기
        it = ids[region][bedCnt][roomType][viewType].insert({ cost, id }).first;
        return cost;
    }

    int reserve(int inTime, int outTime) {
        auto it = reserveList.lower_bound({ inTime, 0 });              // 삽입 가능한 위치 찾기
        if (it != reserveList.begin() && inTime < prev(it, 1)->second) // 직전 일정과 겹치는가?
            return -1;
        if (it != reserveList.end() && it->first < outTime)            // 직후 일정과 겹치는가?
            return -1;
        reserveList.insert({ inTime, outTime });                       // 새로운 일정 삽입
        return id;
    }
}rooms[RLM];
```

Code example


```
void init(int N, int bedCnt[]) {
    ::N = N;
    for (int i = 1; i < HLM; ++i) idsByHotel[i].clear();  // 호텔별 방목록 초기화
    // dis[지역][방갯수][방유형][조망유형] 셋 초기화
    for (int i = 1; i < 11; ++i) for (int j = 2; j < 11; ++j) {
        for (int r = 1; r < 5; ++r) for (int c = 1; c < 5; ++c)
            ids[i][j][r][c].clear();
    }
}

void addRoom(int hotelID, int roomID, int roomInfo[]) {
    idsByHotel[hotelID].push_back(roomID); // 호텔별 방번호 추가
    rooms[roomID].init(roomInfo, roomID);  // 방정보 저장, 분류별 우선순위에 맞게 저장
}
```

Code example


```
int findRoom(int filter[]) {
    int re = filter[2], be = filter[3], rt = filter[4], vt = filter[5];
    for (auto pa : ids[re][be][rt][vt]) {  // 분류별, 우선순위에 따라 일정에 맞는 방 찾기
        if (rooms[pa.second].reserve(filter[0], filter[1]) > 0)
            return pa.second;
    }
    return -1;
}

int riseCosts(int hotelID) {                // hotelID 소속의 방 대여료 인상시키기
    int sum = 0;
    for (int id : idsByHotel[hotelID]) {
        sum += rooms[id].updateCost();
    }
    return sum;
}
```

Thank you.