JooHyun – Lee (comkiwer)

TS주차장관리

Hancom Education Co. Ltd.

Problem

Problem

주차장의 용량과 요금표가 주어진다.

차량이 도착하는 경우, 주차장의 용량 내에서 입차가 되고,
용량을 초과하면 밖에서 대기하게 된다.

차량이 출발하는 경우, 주차장에 주차되어 있었다면 요금표에 따라 주차 요금을 계산한다.

그리고 대기 차량이 있다면 대기 차량 중에서
(총 대기 시간 - 총 주차 시간)이 가장 큰 차량을 주차장에 입차 시킨다.

만약에 (총 대기 시간 - 총 주차 시간)이 가장 큰 차량이 2대 이상이라면,
대기열에 먼저 추가된 차량을 입차 시킨다.

Problem

요금표는 기본 시간, 기본 요금, 단위 시간, 단위 요금으로 구성되어 있다.

주차 시간이 기본 시간이하라면 기본 요금만 부과되고,
기본 시간을 초과하면 단위 시간마다 단위 요금이 부과된다.

이 때 초과한 시간이 단위 시간으로 나누어 떨어지지 않는 경우 올림을 한다.

Problem

예를 들어 요금표가 아래의 [Table 1]과 같은 경우를 살펴보자.

기본 시간	기본 요금	단위 시간	단위 요금
60	5000	20	300

[Table 1]

주차 시간이 50이라면 주차요금은 5000이 되고,
주차 시간이 85라면 5000 + ceil((85 - 60) / 20) x 300 = 5600이 된다.

(여기서 ceil은 올림 함수를 의미한다.)

위와 같이 동작하는 주차 관리 시스템을 구현하여 보자.

※ 아래 API 설명을 참조하여 각 함수를 구현하라.

Problem

void init(int baseTime, int baseFee, int unitTime, int unitFee, int capacity)

각 테스트 케이스의 처음에 호출된다.

주차장이 빈 상태가 된다.

기본 시간은 baseTime, 기본 요금은 baseFee, 단위 시간은 unitTime, 단위 요금은 unitFee이다.

그리고 주차장의 용량은 capacity이다.

Parameters

baseTime: 기본 시간 (60 ≤ baseTime ≤ 180)

baseFee: 기본 요금 (1,000 ≤ baseFee ≤ 20,000)

unitTime: 단위 시간 (20 ≤ unitTime ≤ 120)

unitFee: 단위 요금 (300 ≤ unitFee ≤ 10,000)

capacity: 주차장 용량 (5 ≤ capacity ≤ 200)

Problem

int arrive(int curTime, int carNum)

curTime 시각에 carNum가 도착한다.

carNum가 주차 중이거나 대기 중인 차량 번호로 주어지는 경우는 없다.

주차장의 용량이 남아 있다면 입차 시키고, 그렇지 않다면 대기열에 추가한다.

대기 중인 차량의 개수를 반환한다.

주차장을 방문했던 차량이 다시 방문할 수 있음을 유의하라.

Parameters

curTime: **도착 시각** (1 ≤ curTime ≤ 300,000)

carNum: **차량 번호** (1 ≤ carNum ≤ 1,000,000,000)

Returns

대기 차량의 개수를 반환한다. 대기 차량이 없다면 0을 반환한다.

Problem

int leave(int curTime, int carNum)

curTime 시각에 carNum가 출발한다.

carNum는 주차 중이거나 대기 중인 차량 번호로만 주어진다.

carNum가 주차 중이었다면 주차 요금을 반환한다.

그리고 대기 차량 중에서 (총 대기 시간 - 총 주차 시간)이 가장 큰 차량을 주차장에 입차 시킨다.

만약에 (총 대기 시간 - 총 주차 시간)이 가장 큰 차량이 2대 이상이라면,

대기열에 먼저 추가된 차량을 입차 시킨다.

(대기열에 A 차량, B 차량 순서로 추가된 후에 A 차량이 대기열에서 삭제되고 다시 추가된다면,

B 차량이 A 차량보다 대기열에 먼저 추가된 차량으로 처리됨을 유의하라)

carNum가 대기 중이었다면 대기열에서 삭제하고 -1을 반환한다.

Problem

int leave(int curTime, int carNum) : 계속

...

Parameters

curTime: **출발 시각** (1 ≤ curTime ≤ 300,000)

carNum: **차량 번호** (1 ≤ carNum ≤ 1,000,000,000)

Returns

주차 중이었던 경우, 주차 요금을 반환한다.

대기 중이었던 경우, -1을 반환한다.

Problem

[제약사항]

1. 각 테스트 케이스 시작 시 init() 함수가 호출된다.
2. 각 테스트 케이스에서 curTime 시각은 증가하는 값으로 주어진다.
3. 각 테스트 케이스에서 arrive() 함수의 호출 횟수는 70,000 이하이다.
4. 각 테스트 케이스에서 모든 함수의 호출 횟수 총합은 100,000 이하이다.

Problem analysis

Problem analysis : 예제

(순서 1) 초기에는 주차장과 대기열이 모두 비어 있다.

Order	Function	return	Figure
1	init(60, 5000, 20, 300, 5)		

baseTime : 60

baseFee : 5000

unitTime : 20

unitFee : 300

capacity : 5

Problem analysis : 예제

(순서 2) 10 시각에 200 차량이 도착한다. 주차장에 입차 한다. 대기 차량이 없으므로 0을 반환한다.
(순서 3) 30 시각에 100 차량이 도착한다. 주차장에 입차 한다. 대기 차량이 없으므로 0을 반환한다.
(순서 4) 50 시각에 700 차량이 도착한다. 주차장에 입차 한다. 대기 차량이 없으므로 0을 반환한다.
(순서 5) 80 시각에 600 차량이 도착한다. 주차장에 입차 한다. 대기 차량이 없으므로 0을 반환한다.

Order	Function	return	Figure
2	arrive(10, 200)	0	
3	arrive(30, 100)	0	
4	arrive(50, 700)	0	
5	arrive(80, 600)	0	Fig. 1

Time= 80

Parking
car= 200 (parking time= 70, total parking time= 70, total waiting time = 0)
car= 100 (parking time= 50, total parking time= 50, total waiting time = 0)
car= 700 (parking time= 30, total parking time= 30, total waiting time = 0)
car= 600 (parking time= 0, total parking time= 0, total waiting time = 0)

[Fig. 1]

Problem analysis : 예제

(순서 6) 90 시각에 200 차량이 출발한다. 주차 중이었기 때문에 주차 요금을 계산한다.

주차 요금은 5000 + ceil((80 - 60) / 20) x 300 = 5300이다.

Order	Function	return	Figure
6	leave(90, 200)	5300	Fig. 2

Time= 90

Parking
~~car= 200 (parking time= 80, total parking time= 80, total waiting time = 0)~~
car= 100 (parking time=60, total parking time= 60, total waiting time = 0)
car= 700 (parking time=40, total parking time= 40, total waiting time = 0)
car= 600 (parking time=10, total parking time= 10, total waiting time = 0)

[Fig. 2]

Problem analysis : 예제

(순서 7) 100 시각에 300 차량이 도착한다. 주차장에 입차 한다. 대기 차량이 없으므로 0을 반환한다.

(순서 8) 120 시각에 800 차량이 도착한다. 주차장에 입차 한다. 대기 차량이 없으므로 0을 반환한다.

Order	Function	return	Figure
7	arrive(100, 300)	0	
8	arrive(120, 800)	0	Fig. 3

Time= 120

Parking
car= 100 (parking time= 90, total parking time= 90, total waiting time = 0)
car= 700 (parking time= 70, total parking time= 70, total waiting time = 0)
car= 600 (parking time= 40, total parking time= 40, total waiting time = 0)
car= 300 (parking time= 20, total parking time= 20, total waiting time = 0)
car= 800 (parking time= 0, total parking time= 0, total waiting time = 0)

[Fig. 3]

Problem analysis : 예제

(순서 9) 140 시각에 200 차량이 도착한다. 주차장 용량이 다 찼기 때문에, 대기한다.

대기 차량이 200 차량 1대이므로 1을 반환한다.

Order	Function	return	Figure
9	arrive(140, 200)	1	Fig. 4

Time= 140

Parking
car= 100 (parking time= 110, total parking time= 110, total waiting time = 0)
car= 700 (parking time= 90, total parking time= 90, total waiting time = 0)
car= 600 (parking time= 60, total parking time= 60, total waiting time = 0)
car= 300 (parking time= 40, total parking time= 40, total waiting time = 0)
car= 800 (parking time= 20, total parking time= 20, total waiting time = 0)

Waiting
car= 200 (total parking time= 80, total waiting time = 0)

[Fig. 4]

Problem analysis : 예제

(순서 10) 170 시각에 400 차량이 도착한다. 주차장 용량이 다 찼기 때문에, 대기한다.

대기 차량의 개수로 2를 반환한다.

Order	Function	return	Figure
10	arrive(170, 400)	2	Fig. 5

Time= 170

Parking
car= 100 (parking time= 140, total parking time= 140, total waiting time = 0)
car= 700 (parking time= 120, total parking time= 120, total waiting time = 0)
car= 600 (parking time= 90, total parking time= 90, total waiting time = 0)
car= 300 (parking time= 70, total parking time= 70, total waiting time = 0)
car= 800 (parking time= 50, total parking time= 50, total waiting time = 0)

Waiting
car= 200 (total parking time= 80, total waiting time = 30)
car= 400 (total parking time= 0, total waiting time = 0)

[Fig. 5]

Problem analysis : 예제

(순서 11) 240 시각에 900 차량이 도착한다. 주차장 용량이 다 찼기 때문에, 대기한다.

대기 차량의 개수로 3을 반환한다.

Order	Function	return	Figure
11	arrive(240, 900)	3	Fig. 6

Time= 240

Parking
car= 100 (parking time= 210, total parking time= 210, total waiting time = 0)
car= 700 (parking time= 190, total parking time= 190, total waiting time = 0)
car= 600 (parking time= 160, total parking time= 160, total waiting time = 0)
car= 300 (parking time= 140, total parking time= 140, total waiting time = 0)
car= 800 (parking time= 120, total parking time= 120, total waiting time = 0)

Waiting
car= 200 (total parking time= 80, total waiting time = 100)
car= 400 (total parking time= 0, total waiting time = 70)
car= 900 (total parking time= 0, total waiting time = 0)

[Fig. 6]

Problem analysis : 예제

(순서 12) 300 시각에 300 차량이 출발한다. 주차 중이었기 때문에 주차 요금을 계산한다.
주차 요금은 5000 + ceil((200 - 60) / 20) x 300 = 7100이다.
대기 차량 중에서 (총 대기 시간 - 총 주차 시간)이 가장 큰 400 차량을 주차장에 입차 시킨다.

Order	Function	return	Figure
12	leave(300, 300)	7100	Fig. 7

Time= 300

Parking
car= 100 (parking time= 270, total parking time= 270, total waiting time = 0)
car= 700 (parking time= 250, total parking time= 250, total waiting time = 0)
car= 600 (parking time= 220, total parking time= 220, total waiting time = 0)
~~car= 300 (parking time= 200, total parking time= 200, total waiting time = 0)~~
car= 800 (parking time= 180, total parking time= 180, total waiting time = 0)
car= 400 (parking time= 0, total parking time= 0, total waiting time = 130)

Waiting
car= 200 (total parking time= 80, total waiting time = 160)
~~car= 400 (total parking time= 0, total waiting time = 130)~~
car= 900 (total parking time= 0, total waiting time = 60)

[Fig. 7]

Problem analysis : 예제

(순서 13) 310 시각에 900 차량이 출발한다. 대기 중이었기 때문에 대기열에서 삭제하고 -1을 반환한다.

Order	Function	return	Figure
13	leave(310, 900)	-1	Fig. 8

Time= 310

Parking
car= 100 (parking time= 280, total parking time= 280, total waiting time = 0)
car= 700 (parking time= 260, total parking time= 260, total waiting time = 0)
car= 600 (parking time= 230, total parking time= 230, total waiting time = 0)
car= 800 (parking time= 190, total parking time= 190, total waiting time = 0)
car= 400 (parking time= 10, total parking time= 10, total waiting time = 130)

Waiting
car= 200 (total parking time= 80, total waiting time = 170)
car= 900 (total parking time= 0, total waiting time = 70)

[Fig. 8]

Problem analysis : 예제

(순서 14) 340 시각에 100 차량이 출발한다. 주차 중이었기 때문에 주차 요금을 계산한다.
주차 요금은 5000 + ceil((310 - 60) / 20) x 300 = 8900이다.
대기 차량이 200 차량 1대이므로 200 차량을 주차장에 입차 시킨다.

Order	Function	return	Figure
14	leave(340, 100)	8900	Fig. 9

Time= 340

Parking
~~car= 100 (parking time= 310, total parking time= 310, total waiting time = 0)~~
car= 700 (parking time= 290, total parking time= 290, total waiting time = 0)
car= 600 (parking time= 260, total parking time= 260, total waiting time = 0)
car= 800 (parking time= 220, total parking time= 220, total waiting time = 0)
car= 400 (parking time= 40, total parking time= 40, total waiting time = 130)
car= 200 (parking time= 0, total parking time= 80, total waiting time = 200)

Waiting
~~car= 200 (total parking time= 80, total waiting time = 200)~~

[Fig. 9]

Problem analysis : 예제

(순서 15) 350 시각에 500 차량이 도착한다. 주차장 용량이 다 찼기 때문에, 대기한다.

대기 차량이 500 차량 1대이므로 1을 반환한다.

Order	Function	return	Figure
15	arrive(350, 500)	1	Fig. 10

Time= 350

Parking
car= 700 (parking time= 300, total parking time= 300, total waiting time = 0)
car= 600 (parking time= 270, total parking time= 270, total waiting time = 0)
car= 800 (parking time= 230, total parking time= 230, total waiting time = 0)
car= 400 (parking time= 50, total parking time= 50, total waiting time = 130)
car= 200 (parking time= 10, total parking time= 90, total waiting time = 200)

Waiting
car= 500 (total parking time= 0, total waiting time = 0)

[Fig. 10]

Problem analysis : 예제

(순서 16) 400 시각에 900 차량이 도착한다. 주차장 용량이 다 찼기 때문에, 대기한다.

대기 차량의 개수로 2를 반환한다.

Order	Function	return	Figure
16	arrive(400, 900)	2	Fig. 11

Time= 400

Parking
car= 700 (parking time= 350, total parking time= 350, total waiting time = 0)
car= 600 (parking time= 320, total parking time= 320, total waiting time = 0)
car= 800 (parking time= 280, total parking time= 280, total waiting time = 0)
car= 400 (parking time= 100, total parking time= 100, total waiting time = 130)
car= 200 (parking time= 60, total parking time= 140, total waiting time = 200)

Waiting
car= 500 (total parking time= 0, total waiting time = 50)
car= 900 (total parking time= 0, total waiting time = 70)

[Fig. 11]

Problem analysis : 예제

(순서 17) 420 시각에 200 차량이 출발한다. 주차 중이었기 때문에 주차 요금을 계산한다.
주차 요금은 5000 + ceil((80 - 60) / 20) x 300 = 5300이다.
대기 차량 중에서 (총 대기 시간 - 총 주차 시간)이 가장 큰 900 차량을 주차장에 입차 시킨다.

Order	Function	return	Figure
17	leave(420, 200)	5300	Fig. 12

Time= 420

Parking
car= 700 (parking time= 370, total parking time= 370, total waiting time = 0)
car= 600 (parking time= 340, total parking time= 340, total waiting time = 0)
car= 800 (parking time= 300, total parking time= 300, total waiting time = 0)
car= 400 (parking time= 120, total parking time= 120, total waiting time = 130)
~~car= 200 (parking time= 80, total parking time= 160, total waiting time = 200)~~
car= 900 (parking time= 0, total parking time= 0, total waiting time = 90)

Waiting
car= 500 (total parking time= 0, total waiting time = 70)
~~car= 900 (total parking time= 0, total waiting time = 90)~~

[Fig. 12]

Problem analysis : 예제

(순서 18) 450 시각에 900 차량이 출발한다. 주차 중이었기 때문에 주차 요금을 계산한다.

주차 요금은 5000이다. 대기 차량이 500 차량 1대이므로 500 차량을 주차장에 입차 시킨다.

Order	Function	return	Figure
18	leave(450, 900)	5000	Fig. 13

Time= 450

Parking
car= 700 (parking time= 400, total parking time= 400, total waiting time = 0)
car= 600 (parking time= 370, total parking time= 370, total waiting time = 0)
car= 800 (parking time= 330, total parking time= 330, total waiting time = 0)
car= 400 (parking time= 150, total parking time= 150, total waiting time = 130)
~~car= 900 (parking time= 30, total parking time= 30, total waiting time = 90)~~
car= 500 (parking time= 0, total parking time= 0, total waiting time = 100)

Waiting
~~car= 500 (total parking time= 0, total waiting time = 100)~~

[Fig. 13]

Problem analysis : 제약조건 및 API

- 시간에 흐름에 따른 변화를 적절히 처리해야 하는
  time flow문제이다.

- 주차시스템이 관리하는 모든 차량은 다음 3가지 상태 중 하나이다.
  1. 대기중인 상태 : 대기열에서 총대기시간이 증가중인 상태
  2. 주차중인 상태 : 총주차시간이 증가중인 상태
  3. 주자장을 나간 상태 : 하나의 TC가 실행 중이라면
     총대기시간과, 총주차시간이 증가 또는 소멸되지 않고 유지된다.

- 주차장이 가득 찬 상황에서 주차중인 차량이 주차장을 나가는 것을
  처리하는 것은 단순하다.
- 그런데 빈 자리에 대기중인 차량들 중에 가장 우선순위 높은 차를 주차 시켜야 한다.

Problem analysis : 제약조건 및 API

- 이는 단순한 문제가 아니다.
- 우선순위 조건이 (총대기시간 – 총주차시간)이 가장 큰 차량이기 때문이다.

- 총대기시간은 시간이 흐르며 계속 증가한다.
- 이를 대기중인 모든 차량에 대하여 실시간 업데이트 한다면
  시간 복잡도는 어떻게 될까?
  - arrive()함수가 70,000번 호출 될 수 있고 주차장 용량이 5일수 있으므로
    5대가 주차 중이면 69,995대가 대기중일 수 있다.
  - 함수호출의 총합은 10만 번 가능하므로 주차중인 차량이 나가고
    가장 우선순위 높은 차량을 주차하는 작업을 30000번 실행할 수 있다.
  - 단순히 전체를 탐색한다면 시간복잡도는 O(3만 * 7만 = 21억) 이 된다.

    시간복잡도를 좀 더 줄일 수 있을까?

Solution sketch

Solution sketch

- 차량 번호가 1~10억 이므로 renumbering hash를 사용하여
  1 ~ 70,000 범위의 수로 바꾸어 사용할 수 있다.
  : arrive()함수의 최대 호출 수는 70,000 이다.

  이 경우 unordered_map<int, int> hmap; 이 추천된다.

- 대기차량을 관리하기 위하여 단순 탐색하는 것은 시간이 너무 많이 걸리므로
  우선순위 자료구조 set을 사용하기로 하자. (PQ도 가능)
- 가장 우선순위 높은 차량 하나를 구하는 시간복잡도를 비교해보면 다음과 같다.
  - 단순 전 탐색 : O(70,000)
  - PQ or set : O($\log_2 70{,}000$)이므로 약 O(16~17)

Solution sketch

- set을 사용하기 위해서는 우선순위 기준을 정해 주어야 한다.
- 우선 1순위는 (총대기시간 – 총주차시간) 이 가장 큰 차량이다.

- 총 대기 시간은 시간이 흐름에 따라 증가한다.
- 대기열에 있는 모든 차량은 똑같이 시간이 흐르므로 실시간 업데이트가 필요하지 않다.

- 그런데 새로운 차량이 대기열에 들어오는 경우는 상황이 달라진다.
  기존에 대기열에 있던 차량들은 현시점을 적용하지 않은 상태이므로
  대기열의 모든 차량의 총대기시간을 업데이트 해야 한다.

  그런데 이렇게 하면 우선순위 자료를 사용하는 이점이 사라진다.
  어떻게 할 것인가?

Solution sketch

- 두 차량 A, B의 우선순위 를 비교하기 위하여
  두 차량 A, B의 (총대기시간 – 총주차시간) 를 구해보자.
  흐르는 현재 시각 Tick시점에서 구해보면 다음과 같을 것이다.
  - A의 총대기시간 – A의 총주차시간 =
    Tick – A의 대기시작시각 + A의 대기시작시각 이전까지 대기시간의 합 – A의 총주차시간
  - B의 총대기시간 – B의 총주차시간=
    Tick - B의 대기시작시각 + B의 대기시작시각 이전까지 대기시간의 합 - B의 총주차시간

- 두 식에서 흐르는 현재 시각 Tick은 같음을 알 수 있다.
  이는 모든 차량에 대하여 Tick은 제외하고 비교해도 문제가 되지 않음을 의미한다.

Solution sketch

- 이제 우리는 우선순위 대기열의 모든 차량에 대하여
  총대기시간 – 총주차시간을
  대기시작시각 이전까지 대기시간의 합 – 총주차시간 – 대기시작시각 으로
  바꾸어 처리할 수 있다.

- 대기시작시각 이전까지 대기시간의 합 – 총주차시간을
  `int wpTime[LM];` 으로 관리하기로 하자.
  (wpTime : waiting parking time)

- 대기시작시각은 `int st[LM];  // start time`
  우선 1순위는 `int prior[LM]; // wpTime[id] - st[id]`
  로 관리하자.

Solution sketch

- 우선순위를 다루는 set을 다음과 같이 정의 할 수 있다.

```
struct Cmp {
    bool operator()(const int a, const int b)const {
        return prior[a] != prior[b] ? prior[a] > prior[b]:st[a] < st[b];
    }
};
set<int, Cmp> myset;                // 대기중인 차량을 우선순위에 맞게 유지
set<int, Cmp>::iterator its[LM]; // 실시간 업데이트 효율을 높이기 위하여: 삭제 시 사용
```

- 이제 각 함수 별 할 일을 정의해 보자.

Solution sketch

void init(int baseTime, int baseFee, int unitTime, int unitFee, int capacity)

- 기본시간, 기본요금, 단위시간, 단위요금, 주차장용량을 전역변수에 저장한다.
- renumbering_id를 위한 idCnt = 0
- 주차대수 parkingCnt = 0으로 초기화 한다.
- hmap.clear(); // renumbering id 해시 테이블
- myset.clear();

Solution sketch

int arrive(int curTime, int carNum)

- carNum에 대한 renumbering_id를 구한다.
  이를 int cid라고 하자.
- cid의 대기 또는 주차 시작시간 st를 설정한다.
  st[cid] = curTime;
- 주차가 가능하다면 주차한다.
  주차중임을 표시한다.
- 불가능한 경우
  1. 우선 1순위를 구하여 저장한다. prior[cid] = wpTime[cid] – curTime;
  2. myset에 cid를 추가하고 그 위치를 its[cid]에 저장한다.

- 대기 대수인 myset.size()를 반환하고 함수를 종료한다.

Solution sketch

int leave(int curTime, int carNum)

- carNum의 재설정 아이디 cid를 구한다. `int cid = hmap[carNum];`
- cid가 주차중이라면
  1. 주차대수감소 및 주차 시간 정산
  2. 주차시간계산하기 : 소수점 이하 올림 하여 계산
  3. 대기중인 차량이 있다면 가장 우선순위 높은 차량을 주차 처리하기
     1) 주차대수 증가
     2) 대기목록에서 제거
     3) 대기시간정산, 주차 시작시간 설정
     4) 주차중임을 표시
- 대기중이라면
  1. 대기시간 업데이트
  2. 대기목록에서 제거
  3. -1반환

Solution sketch

[Summary]

- 시간의 흐름에 따라 처리하는 time flow문제이다.
- 시간의 흐름을 이용하여 우선순위를 결정하는 경우
  효율적으로 처리할 수 있어야 한다.

- renumbering id를 위하여 unordered_map을 이용할 수 있다.
- 우선순위 자료구조를 문제해결에 사용할 수 있다.

[One more step]

- lazy update방법으로 문제를 해결할 수 있다.
- user index PQ를 이용한 real time update로 문제를 해결할 수 있다.

Code example1

Code example1

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

const int LM = 100005;
const int INF = (int)2e9;
using piii = pair<pair<int, int>, int>;
priority_queue<piii> pq;
//baseTime, baseFee, unitTime, unitFee, capacity
int bt, bf, ut, uf, capa;
int wpTime[LM], st[LM], prior[LM]; // wpTime = waitingTime - parkingTime
int idCnt, parkingCnt, waitingCnt;
unordered_map<int, int> hmap;

void init(int baseTime, int baseFee, int unitTime, int unitFee, int capacity) {
    bt = baseTime, bf = baseFee, ut = unitTime, uf = unitFee, capa = capacity;
    idCnt = parkingCnt = waitingCnt = 0;
    hmap.clear();
    pq = {};
}
```

Code example1

```
int arrive(int mTime, int carNum) {
    auto it = hmap.find(carNum);
    if (it == hmap.end()) {
        it = hmap.insert({ carNum, ++idCnt }).first;
        wpTime[idCnt] = 0;
    }
    int cid = it->second;
    st[cid] = mTime;       // 주차시작시각 또는 대기시작시각
    if (parkingCnt < capa) {
        parkingCnt++;      // 주차 대수 증가
        prior[cid] = INF; // 주차 중임을 표시
    }
    else {
        waitingCnt++;      // 대기 대수 증가
        prior[cid] = wpTime[cid] - mTime;        // 대기 중임을 표시
        pq.push({ {prior[cid], -mTime}, cid }); // 우선순위 대기열에 추가
    }
    return waitingCnt;     // 현재 대기 자동차 대수 반환
}
```

Code example1

```
int leave(int mTime, int carNum) {
    int cid = hmap[carNum];
    if (prior[cid] == INF) { // 주차중인 자동차
        parkingCnt--;
        int parkingTime = mTime - st[cid];
        wpTime[cid] -= parkingTime;          // 주차시간은 빼준다.

        int fee = bf;        // 주차요금계산
        if (parkingTime > bt)
            fee += (parkingTime - bt + ut - 1) / ut * uf;

        if (waitingCnt) {    // 대기 중인 자동차가 있다면
            waitingCnt--, parkingCnt++;
            while (prior[pq.top().second] != pq.top().first.first)
                pq.pop();    // invalid 데이터 제거
            cid = pq.top().second;
            wpTime[cid] += mTime - st[cid];  // 대기시간은 더한다.
            st[cid] = mTime; // 주차 시작시각
            prior[cid] = INF;
        }
        return fee;
    }

    waitingCnt--;
    wpTime[cid] += mTime - st[cid];          // 대기시간은 더한다.
    prior[cid] = -INF; // pq유효성 검증을 위하여: 대기중이 아님을 표시
    return -1;
}
```

Code example2

Code example2

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

const int LM = 100005;
const int INF = (int)2e9;
//baseTime, baseFee, unitTime, unitFee, capacity
int bt, bf, ut, uf, capa;
int wpTime[LM];  // waitingTime - parkingTime
int st[LM];      // 대기(2우선순위) 또는 주차를 시작하는 시각
int prior[LM];   // 대기(1우선순위), 주차(INF로 주차표시)
int parkingCnt;  // 주차중인 차량수

int idCnt;       // renumbering id 개수
unordered_map<int, int> hmap;  // <int(ID), int(renumbering id)>

struct Cmp {
    bool operator()(const int a, const int b)const {
        return prior[a] != prior[b] ? prior[a] > prior[b]:st[a] < st[b];
    }
};
set<int, Cmp> myset;
set<int, Cmp>::iterator its[LM]; // 삭제등 참조의 효율향상 : 실행속도 2배 이상 빨라짐
```

Code example2

```
void init(int baseTime, int baseFee, int unitTime, int unitFee, int capacity) {
    bt = baseTime, bf = baseFee, ut = unitTime, uf = unitFee, capa = capacity;

    for (int i = 0; i <= idCnt; ++i) wpTime[i] = 0;
    idCnt = parkingCnt = 0;
    hmap.clear();
    myset.clear();
}

int arrive(int mTime, int carNum) {
    auto it = hmap.find(carNum);
    if (it == hmap.end()) {  // 새로운 차량이라면
        it = hmap.insert({ carNum, ++idCnt }).first;
    }
    int cid = it->second;     // carNum의 renumbering id
    st[cid] = mTime;          // 대기 시작 또는 주차시작 시각 기록
    if (parkingCnt < capa) { // 주차공간이 있다면
        parkingCnt++;         // 주차 대수 증가
        prior[cid] = INF;     // 주차중임을 표시
    }
    else {                    // 주차공간이 없는 경우
        prior[cid] = wpTime[cid] - mTime;   // 1우선순위
        its[cid] = myset.insert(cid).first; // 저장위치 백업
    }
    return myset.size();      // 주차대수 반환
}
```

Code example2

```
int leave(int mTime, int carNum) {
    int cid = hmap[carNum];
    if (prior[cid] == INF) {  // 주차중인 자동차
        parkingCnt--;         // 주차대수 감소
        int parkingTime = mTime - st[cid];  // 주차시간
        wpTime[cid] -= parkingTime;         // 주차시간은 빼준다.

        // 주차요금계산
        int fee = bf;           // 기본요금계산
        parkingTime -= bt;      // 기본시간 제외시키기
        if (parkingTime > 0) { // 남은 시간에 대하여 단위시간으로 계산
            fee += (parkingTime + ut - 1) / ut * uf; // 올림 계산
        }
        if (!myset.empty()) {
            parkingCnt++;                   // 주차 대수 증가
            cid = *myset.begin();           // 가장 우선순위 높은 차량
            myset.erase(its[cid]);          // 대기 목록에서 제거하기
            wpTime[cid] += mTime - st[cid]; // 대기 시간은 더해준다.
            st[cid] = mTime;                // 주차 시작 시각 기록
            prior[cid] = INF;               // 주차중임을 표시
        }
        return fee;
    }

    wpTime[cid] += mTime - st[cid];         // 대기 시간은 더해준다.
    myset.erase(its[cid]);                  // 대기 목록에서 제거하기
    return -1;
}
```

Code example3

Code example3

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

const int LM = 100005;
const int INF = (int)2e9;
//baseTime, baseFee, unitTime, unitFee, capacity, curTime
int bt, bf, ut, uf, capa;
int parkingCnt; // 주차중인 차량 대수
int wpTime[LM]; // wpTime = waitingTime - parkingTime
                // 한 TC에서 차량별 wpTime[]은 계속 누적된다.
int st[LM];     // st = startTime(wait or park)
int prior[LM];  // prior = [currentTime +] waingTime - parkingTime - st;
                // currentTime(흘러가는 매 시각)은 모든 차에 동일하므로 생략 가능.

int idCnt;
unordered_map<int, int> hmap;  // id renumbering hash
```

Code example3

```
struct PQ {
    int hp[LM], index[LM], hn;
    bool cmp(int a, int b) {
        return prior[a] != prior[b] ? prior[a] > prior[b]:st[a] < st[b];
    }
    void clear() { hn = 0; }
    bool empty() { return hn == 0; }
    int size() { return hn; }
    int top() { return hp[1]; }
    void push(int id) { up(id, ++hn); }
    void pop() { erase(top()); }
    void update(int id) {
        if (index[id] == 0) return;
        up(id, index[id]), down(id, index[id]);
    }
```

Code example3

```
    void erase(int tg) {
        if (index[tg] == 0) return;
        int id = hp[hn--];
        index[id] = index[tg], index[tg] = 0;
        update(id);
    }
    void up(int id, int c) {
        for (; c > 1 && cmp(id, hp[c >> 1]); c >>= 1)
            hp[c] = hp[c >> 1], index[hp[c]] = c;
        hp[c] = id, index[id] = c;
    }
    void down(int id, int c) {
        c <<= 1;
        for (; c <= hn && cmp(hp[c += (c < hn && cmp(hp[c + 1], hp[c]))], id); c <<= 1)
            hp[c >> 1] = hp[c], index[hp[c]] = c >> 1;
        hp[c >> 1] = id, index[id] = c >> 1;
    }
}pq;
```

Code example3

```
void init(int baseTime, int baseFee, int unitTime, int unitFee, int capacity) {
    bt = baseTime, bf = baseFee, ut = unitTime, uf = unitFee, capa = capacity;
    idCnt = parkingCnt = 0;
    hmap.clear();
    pq.clear();
}

int arrive(int curTime, int carNum) {
    auto it = hmap.find(carNum);
    if (it == hmap.end()) {
        it = hmap.insert({ carNum, ++idCnt }).first;
        wpTime[idCnt] = 0;                  // 처음등장한 경우 0으로 초기화
    }
    int id = it->second;                    // renumbering id
    st[id] = curTime;                       // 주차또는대기 시작시각
    if (parkingCnt < capa) {
        parkingCnt++;                       // 주차대수 증가
        prior[id] = INF;                    // 주차중임을 표시
    }
    else {
        prior[id] = wpTime[id] - curTime;   // 대기를 위한 first 우선순위
        pq.push(id);                        // 대기 목록에 추가
    }
    return pq.size();
}
```

Code example3

```
int leave(int curTime, int carNum) {
    int id = hmap[carNum];
    if (prior[id] == INF) {          // 주차중인 자동차라면
        parkingCnt--;
        int parkingTime = curTime - st[id];
        wpTime[id] -= parkingTime;  // 주차시간은 빼준다.
        int fee = bf;
        if (parkingTime > bt)
            fee += (parkingTime - bt + ut - 1) / ut * uf;
        if (!pq.empty()) {           // 대기중인 차량이 있다면
            parkingCnt++;
            id = pq.top(), pq.pop();
            wpTime[id] += curTime - st[id]; // 대기시간은 더해준다.
            st[id] = curTime; // 주차 시작시각
            prior[id] = INF;  // 주차 중임을 표시
        }
        return fee;
    }

    wpTime[id] += curTime - st[id]; // 대기시간은 더해준다.
    //prior[id] = -INF;  // 운행중임을 표시
    pq.erase(id);
    return -1;
}
```

Thank you.