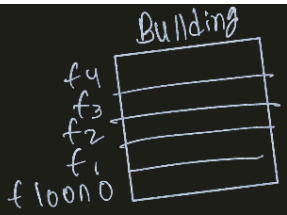
Rough flow



Request clarification

* How many lifts to consider?

  n

* lift dispatch algorithm

  algorithm:

  lift1 → odd

  lift2 → even

  or

  lift1 → 1 to 3

  lift2 → 4 to 6

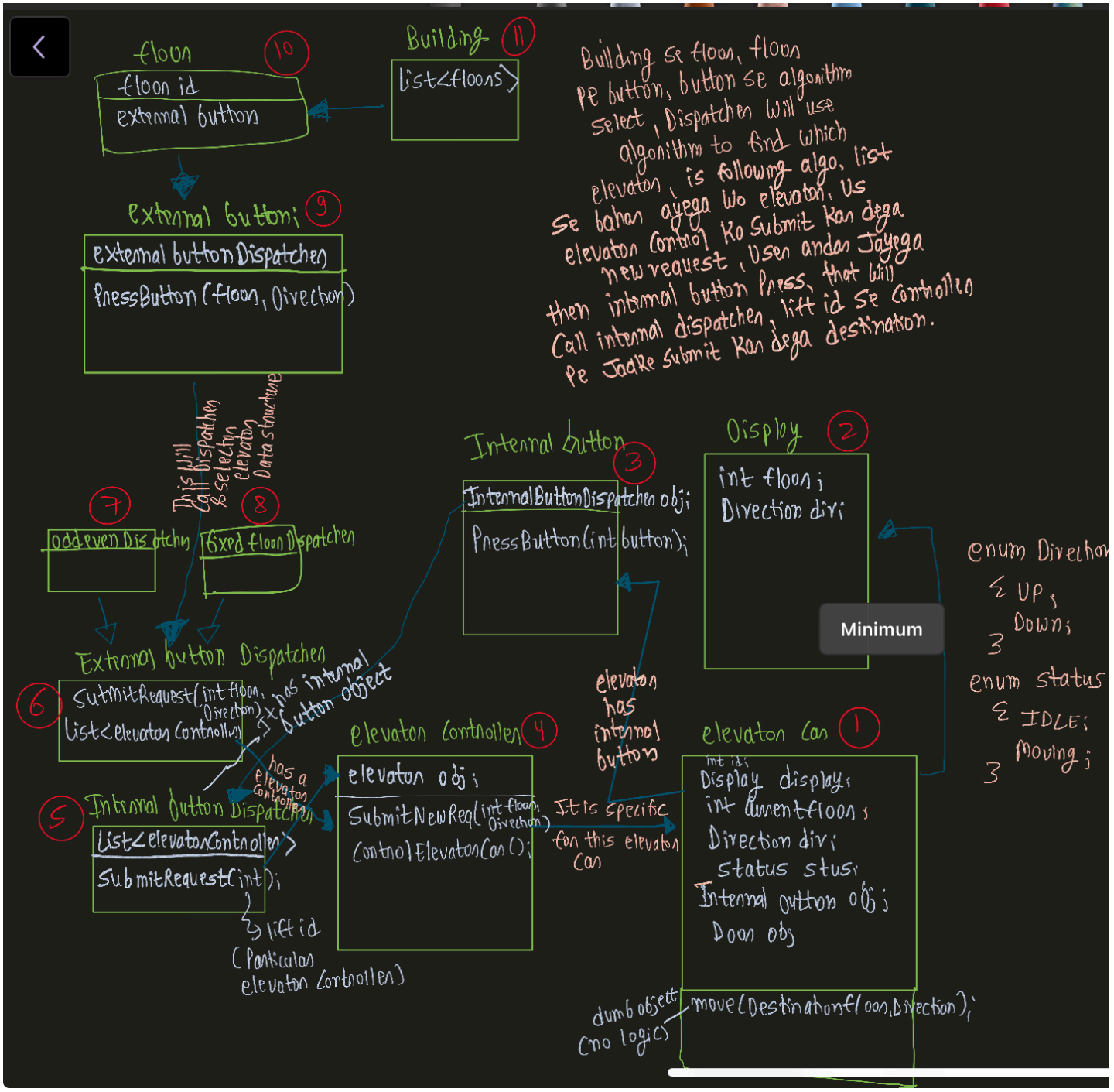
* odd/even
* fixed floor
* Minimum seek time

  (Jo sabse pass ho usey K)

objects

* building
* floor
* External button
* Elevator Car
  * Current floor
  * Display
  * Direction { UP, Down
  * Status { Moving, idle
  * internal button
  * Doors
* Display
* internal button
* doors

Minimum

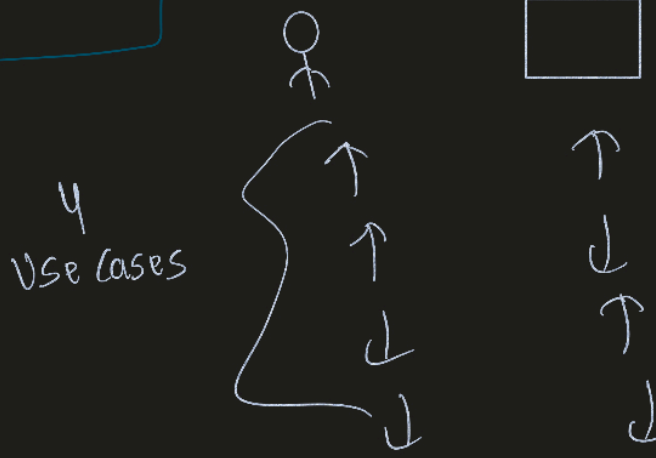

floor
floor id
external button
10
Building
11
list<floors>
Building se floor, floor pe button, button se algorithm select, Dispatcher will use algorithm to find which elevator, is following algo, list se bahar ayega wo elevator, us elevator control ko submit kar dega new request, user andar jayega then internal button press, that will call internal dispatcher, lift id se controller pe jake submit kar dega destination.
external button;
9
external button Dispatcher
PressButton (floor, Direction)
This will Dispatcher selection elevator Data structure
7
odd even Dispatcher
8
fixed floor Dispatcher
Internal button
3
InternalButtonDispatcher obj;
PressButton(int button);
Display
2
int floor;
Direction dir;
Minimum
enum Direction
{ UP;
Down;
}
enum status
{ IDLE;
moving;
}
External button Dispatcher
6
SubmitRequest(int floor, Direction)
list<elevator controllers>
has internal button object
has a elevator controller
elevator has internal button
elevator controller
4
elevator obj;
SubmitNewReq(int floor, Direction)
controlElevatorCar();
It is specific for this elevator car
elevator car
1
int id;
Display display;
int currentfloor;
Direction dir;
status stus;
Internal button obj;
Door obj
Internal button Dispatcher
5
List<elevatorController>
SubmitRequest(int);
lift id
(Particular elevator controller)
dumb object (no logic)
move(Destinationfloor, Direction);

algorithm

external Dispatcher
(odd even, fixed, --- etc)

elevator Controller

USE CASE



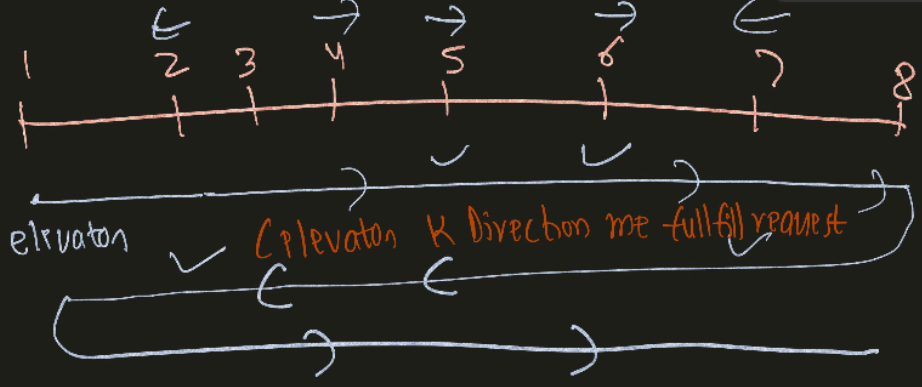
4
Use cases

(User want to go UP & lift is going up)

SCAN algorithm (elevator algorithm)

* Suppose We have 8 floor buildi

Minimum

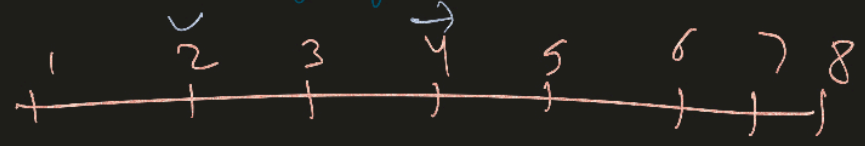


(elevator K Direction me fullfill request

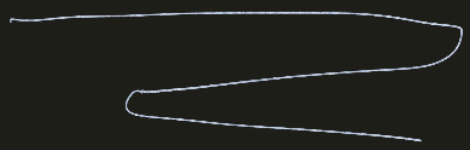
Disadvantage: Jab aage request nahi hai toh aage kyu Jaana hai.

Suppose 5 k baad request nahi hai to 5 K aage kyu Jaana hai

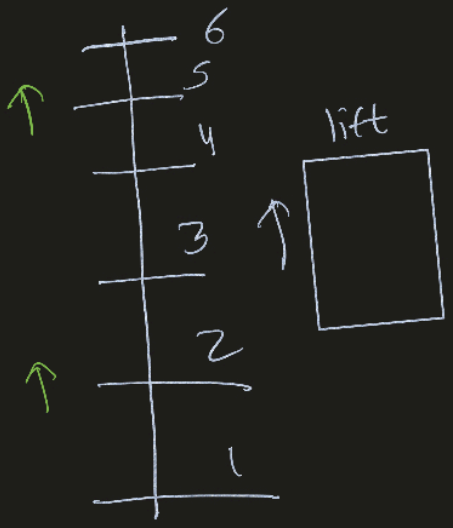
LOOK algorithm



(Jab lift aage badh rahi toh ek step aage dekhega, ki aage koi request hai, agar nahi hai toh wahi se mud jao)



To implement LOOK, what Data Structure needed?



(lift upar ja rahi & neeche se request aaya, upar jaane ko toh fulfill nahi hoga, upar se aayega tab hi hoga)

(Upar jaate waqt ascending order me request accept karega neeche jaate waqt descending order me)

Take 2 Priority Queue:

1. Min Priority Queue for Up (Current Postion se upar)
2. Max Priority Queue for down (Current Position se neeche)
3. Queue / array for Keeping Pending Job
   (Upar jaate waqt jo neeche hai)
   → Opposite Direction Waale

neeche Jaate Waqt (Desc)
order me)

Take 2 Priority Queue:

1. Min Priority Queue for Up (Current Postion se Upar)
2. Max Priority Queue for down (Current Position se neeche)
3. Queue / array for Keeping Pending Job
   - (Upar Jaate Waqt Jo neeche hai)
   - → Opposite Direction Waale

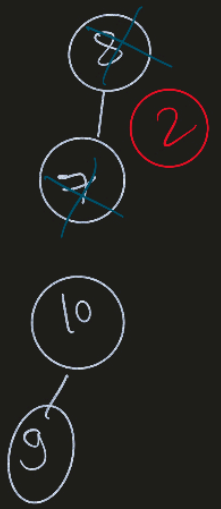
Current *

Up: 3, 6, 4

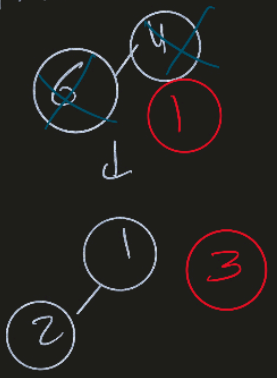
down: 7, 8

Minimum

Max PQ



Min PQ



Pending