Câu 1 :

Bài toán đặt ra có N tầng và M thang máy .Đầu ra : (nhật ký bảng điều khiển, bất kỳ GUI hoặc hoạt động nào) có thể cho thấy rằng thang máy có thể nhận yêu cầu và xử lý để phản hồi các yêu cầu đó

Ý tưởng :

Thang máy có các trang thái :(-1 đi xuống,0 đứng yên,1 đi lên).

Giải thuật:

- Hàm chọn thang máy :

- thang cùng chiều
  + Thang đi lên, khách đi lên và vị trí khách khác tầng trên cùng Khoảng cách thang nào tới vị trí của khách nhỏ hơn sẽ được chọn
  + Các trường hợp còn lại, thang được chọn là thang có khoảng cách giữa tầng cuối cùng với vị trí khách nhỏ hơn
- thang ngược chiều
  + Khách đi lên, vị trí khách khác tầng trên cùng thang đi lên sẽ được chọn
  + Khách đi xuống, vị trí khách khác tầng dưới cùng, thang được chọn là thang có khoảng cách giữa tầng cuối cùng(tầng trên cùng hoặc dưới cùng) đến vị trí khách nhỏ hơn
  * thang đứng im, thang chuyển động
  + Nếu thang máy chuyển động lên, khách đi lên, thang máy đó được chọn, nếu không thang còn lại sẽ được chọn

  Giải thuật 2:
  Thang máy có 2 kiểu gọi
  Kiểu 1 là trên đường thang máy di chuyển, dừng lại đón khách
  Kiểu 2 là trên đường thang máy di chuyển, không dừng lại đón khách, đến tầng cuối cùng, quay lại đón khách
  Hàm gọi thang máy đến
  - Nếu thang được gọi theo kiểu 1 và có vị trí tầng khác vị trí khách

- Vị trí thang lớn hơn vị trí khách
for (i = vi tri khách + 1; i<= vị trí thang máy; i++) hướng đi của thang là đi xuống(-1)
- Vị trí thang nhỏ hơn vị trí khách
for (i = vị trí thang máy; i<=vị trí khách-1; i++) hướng đi thang máy là đi lên
- Xác định lại điểm cuối
n=fabs(tầng cuối – vị trí thang máy)
m= fabs(vị trí thang máy – vị trí khách)
nếu m>n tầng cuối = vi trí khách

- vấn đề tối ưu hiệu quả sử dụng

- số lượng thang
+ Đặt ($\frac{m}{2}$ -1) thang ở vị trí tầng $\frac{n}{2}$ và $\frac{m}{2}$ +1 thang ở tầng 0 (có thể dung 1 thang riêng để chuyển đồ)
+ sử dụng nhỏ hơn hoặc bằng ($\frac{m}{2}$ - $\frac{m}{2}+2$) đi từ tầng 0 lên tầng 5
+ sử dụng nhỏ hơn hoặc bằng ($\frac{m}{2}$ - $\frac{m}{2}+2$) đi từ tầng $\frac{n}{2}$ lên tầng n
+ các thanh còn lại dùng để đi tất cả các tầng .
- vấn đề tối ưu lượng người .
+ ví dụ công năng sử dụng buổi sáng (từ ?:00h - ?:00h) lượng người đi chuyển lên hoặc xuống nhiều có thể huy động ($\frac{m}{2}$ - a) thang ở vị trí tầng $\frac{n}{2}$ xuống tầng 0 (a tùy vào lượng người di chuyển lên và xuống)

. chức năng ngoài thang
INPUT
Nút lên
Nút xuống

PROCESS
Đọc dữ liệu đầu vào
Đọc dữ liệu đầu vào

OUTPUT
Cấu trúc khach.vitri
Cấu trúc khach.vitri

Chức năng chọn thang gần nhất
INPUT
Thang 1,....., thang m

PROCESS
Chọn thang gần nhất

. chức năng chọn đường thang đến
INPUT
PROCESS
Thang 1
Thang 2

Chọn đường lên hay xuống

OUTPUT
Thang 1 .... thang m

OUTPUT
Thành công hoặc thất bại
Thành công hoặc thất bại

. chức năng nhập bên trong thang máy

INPUT
Phím bấm tầng

PROCESS
Đọc dữ liệu

OUTPUT
Cấu trúc khach.tangden

- chương trình
  Các hàm có trong chương trình:
  int lenh_goi_thang();
  void lenh_goi_cat_ngang();
  void dieu_thang(thang_may tm, int tang);
  int lenh_trong_thang(thang_may tm);
  void do_lenh_trong(thang_may tm,int tang_dich);
  void xoa_lenh_goi(int n);
  void xoa_lenh_trong(thang_may tm, int n);
  int abi(int a);
  void nhap_lenh_dieu_khien(thang_may tm);
  void gioi_thieu();

Từ số lương output đếm số lượng trạng thái -1 và 1 từng thời điểm cao điểm để tối ưu các thang máy trong khung giờ cao điểm

2. problem 2

hình ảnh dưới dạng ma trận NxN của pixel (mỗi pixel là số). Viết chương trình quay 90 độ hình đó? Bạn có thể làm điều đó tại chỗ mà không cần sử dụng bất kỳ ma trận tạm thời nào không?

Kí hiệu ma trận vuông NxN là R là một ma trận vuông biểu thị phép quay (ma trận quay) và V là một vecto cột mô tả vị trí của 1 điểm trong không gian., tích RV là một vecto cột khác mô tả vị trí của điểm đó sau phép quay đó . Nếu v là một vecto hang , có thể thu được phép biến đổi tương tự bằng cách sử dụng VR^T với R^T là ma trận chuyển vị của R

Chương trình :

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
```

```c
#include <memory.h>
#define N 100
#define M 100

void ImportData(int[][M], int, int, char*);
void PrintMatrix(int[][M], int, int, char);
void Rotate90(int[][M], int[][M], int, int, int*, int*);


void main()
{
    int A[N][M];
    int B[N][M];
    int aRow, aCol;
    int bRow, bCol;

    printf("\nNumber of row: ");
    scanf("%d", &aRow);
    printf("\nNumber of column: ");
    scanf("%d", &aCol);

    ImportData(A, aRow, aCol, "A");
    PrintMatrix(A, aRow, aCol, 'A');


    Rotate90(A, B, aRow, aCol, &bRow, &bCol);

    printf("\n90 degree rotated matrix\n");
    PrintMatrix(B, bRow, bCol, 'B');

    getch();
}


void Rotate90(int A[][M], int B[][M], int aRow, int aCol,
int* bRow, int* bCol)
{
    int iARow, iACol;
```

```
    *bRow = aCol;
    *bCol = aRow;

    for (iACol = aCol-1; iACol >= 0; iACol--)
    {
        for (iARow = 0; iARow < aRow; iARow++)
        {
            B[aCol - 1 - iACol][iARow] = A[iARow][iACol];
        }
    }
}


void ImportData(int Matrix[][M], int nRow, int nCol, char*
nameMatrix)
{
    int iRow, iCol;
    for (iRow = 0; iRow < nRow; iRow++)
        for (iCol = 0; iCol < nCol; iCol++)
        {
            printf("\n%s[%d][%d] = ", nameMatrix, iRow,
iCol);
            scanf("%d", &Matrix[iRow][iCol]);
        }
}

void PrintMatrix(int Matrix[][M], int nRow, int nCol, char
name)
{
    int iRow, iCol;
    printf("\n%c = ", name);
    for (iRow = 0; iRow < nRow; iRow++)
    {
        printf("\n");
        for (iCol = 0; iCol < nCol; iCol++)
        {
            printf("\t%d\t", Matrix[iRow][iCol]);
        }
    }
```

}

- Bạn có thể làm điều đó tại chỗ mà không cần sử dụng bất kỳ ma trận tạm thời nào không?
  Trả lời Không thể : vì phép quay ma trận dựa vào biến đổi ma trận và nhân ma trận với nhau