**PBL ON DAA**

**Mid –Term Project Report**

**ON**

**Synchonous Elevator**

**Submitted By**

|  |
| --- |
| ADARSH MISHRA |
| 1513105013 |
|  |

***Under the guidance of***

**Ms. MONIKA YADAV**

Faculty of CSE

School of Computing Science & Engineering,



**School Of Computing Science and Engineering**

**Galgotias University**

Plot No.2, Sector 17-A  
Yamuna Expressway,  
Greater Noida, Gautam Buddh Nagar,  
Uttar Pradesh, India

**November- 2017**

**School of Computing Science and Engineering**

**Galgotias University**

**PBL Mid –Term Project Report -2(2016-17)**

**Project Title: Synchonous Elevator**

Nowadays, in large buildings, the conventional elevators have higher waiting time, higher traveling time and useless calling of multiple lifts on same floor. So there is a need to develop an algorithm which reduces the above problems. We have developed an algorithm which reduces the waiting and travelling time in elevators.

The keypad of the elevator is outside the lift in the lobby of the building. All the elevators on a particular floor of the building have a common keypad at the entrance of lobby. When a person presses his desired floor on the keypad, the system assigns a lift to him( which is on that floor or nearer to the floor).

***Abstract:***

This project consist of a basic algorithm which manages the lift operation. This algorithm consists of basic algorithmic implementation like optimized approach to the naïve approach.

Dissimilar Assignment of nodes :

In this we have used an algorithm which is analogous to graph coloring but same as graph coloring. No two lift can be assigned to the same floor. According to graph coloring algorithm, no two adjacent vertices can have same color. But here we have twisted the algorithm a little bit such that rather than adjacent nodes we have changed it for all nodes. Means, no two lift can be assigned to same floor.

Windows header file:

Sleep function

Suspends the execution of the current thread until the time-out interval elapses.

To enter an alertable wait state, use the [SleepEx](https://msdn.microsoft.com/en-us/library/windows/desktop/ms686307(v=vs.85).aspx) function.

Syntax

VOID WINAPI Sleep(

\_In\_ DWORD dwMilliseconds

);

Parameters

*dwMilliseconds*

The time interval for which execution is to be suspended, in milliseconds.

A value of zero causes the thread to relinquish the remainder of its time slice to any other thread that is ready to run. If there are no other threads ready to run, the function returns immediately, and the thread continues execution.

Windows XP:  A value of zero causes the thread to relinquish the remainder of its time slice to any other thread of equal priority that is ready to run. If there are no other threads of equal priority ready to run, the function returns immediately, and the thread continues execution. This behavior changed starting with Windows Server 2003.

A value of INFINITE indicates that the suspension should not time out.

Return value

This function does not return a value.

Tail Recursion:

We have used the recursive calls to the the **call( )** functions without any implicit stack formation. This means that there is no statement after the the recursive call of the function. Thus we have also minimised the space complexity of the program.

Linear Search :

In the **findlift( )** we have used linear search for finding if lift is in use at use on some floor or not.

Linear is the most basic searching algorithm. It basically uses one loop and matches each element of the array with the value to be found.

**Introduction:**

Background: This project idea came from the hostel lift chaos. After observing that two student used to call two different lifts from the same floor whereas those who are waiting for a single lift on some other floor has to suffer, thus we thought to develop an algorithm which could solve this problem.

This lift algorithm is an efficient and and ensures optimized utilization of resources available. It prevents multiple lifts beings at same place thus affecting the lift availability at other places.

We can use this algorithm to improve the lifts which we use in our day to day life. Atleast it will save a little of waiting time which is spent while calling for lift. Especially in more chaotic places like Students Hostel, Metro station etc.

**Problem Statement:**

Write a simple program that controls an elevator in a building. The  
building has 10 floors (0--9) and 2 elevators. The floors are stored in a linear list (may be in Array) and the elevators cover each floor.

**Input/ Output:**

First of all at the time of execution of program it will ask for calling from which floor and also which lift you are calling.

After entering the data it will ask whether another call is made or to release the lift.

After every option you will get a lift call proposal.

A sample input output procedure is mentioned below :

Do yo want Call lift 0-No 1-Yes : 1

Calling from floor(0-9) and lift-(1 or 2) :- 3 1

Please wait LIFT - 1 is coming.....

Lift reached

Call Lift 0-yes Release Lift 1-yes

0

Calling from floor(0-9) and lift-(1 or 2) :- 2 2

Please wait LIFT - 2 is coming.....

Lift reached

Call Lift 0-yes Release Lift 1-yes

**Modules and their interaction**

Failed

**CHECK**

success

**RELEASE**

**ASSIGN**

**CALL**

Call() function is called and it first checks whether the lifts are in a condition to receive the calls by calling the check() function.

If check() function return true the the algorithm proceeds further and assigns the lift by calling the assign() finction, but if check () function returns false then it asks again for call after displaying some message for failure.

Assign() function just assigns a lift to a particular floor . And after assigning it ask to release the lift or to call from another user.

Release function relase the lift from current floor and makes it available for other users to be called.

**Algorithm:**

#include <iostream>

#include <windows.h>

using namespace std;

bool findlift(int A[], int lift)

{

for (int i = 0; i < 10; i++)

if (A[i] == lift)

return false;

return true;

}

bool check(int f, int A[], int lift)

{

if (A[f] == 0 && findlift(A, lift))

return true;

return false;

}

int call(int floor, int arr[], int lift)

{

if (check(floor, arr, lift) == true) {

cout << "\nPlease wait LIFT - " << lift << " is coming.....\n";

Sleep(3000);

cout << "\nLift reached\n";

arr[floor] = lift;

cout << "\nCall Lift 0-yes \tRelease Lift 1-yes \n";

int resp = 99;

cin >> resp;

if (resp == 1) {

int resp;

arr[floor] = 0;

cout << "\nLift-" << lift << " Releases\n";

cout << "\nCall Lift 1-yes";

cin >> resp;

if (resp == 1) {

cout << "\nCalling from floor(0-9) and lift-(1 or 2) :- ";

cin >> floor;

call(floor, arr, lift);

}

}

else if (resp == 0) {

cout << "\nCalling from floor(0-9) and lift-(1 or 2) :- ";

cin >> floor >> lift;

call(floor, arr, lift);

}

}

else {

cout << "\nOther lift is on your floor OR Lift is in use please call again \n";

cout << "\nRelease lift 1-yes : \t Call lift yes-2";

int tres = 0;

cin >> tres;

if (tres == 1) {

int resp;

arr[floor] = 0;

cout << "\nLift-" << lift << " Releases\n";

cout << "\nCall Lift 1-yes";

cin >> resp;

if (resp == 1) {

cout << "\nCalling from floor(0-9) and lift-(1 or 2) :- ";

cin >> floor;

call(floor, arr, lift);

}

}

}

}

int main()

{

int arr[10] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };

int lift\_1 = 1, lift\_2 = 2, res = 1, floor, lift;

while (1) {

cout << "Do yo want Call lift 0-No 1-Yes : ";

cin >> res;

if (res == 1) {

cout << "Calling from floor(0-9) and lift-(1 or 2) :- ";

cin >> floor >> lift;

call(floor, arr, lift);

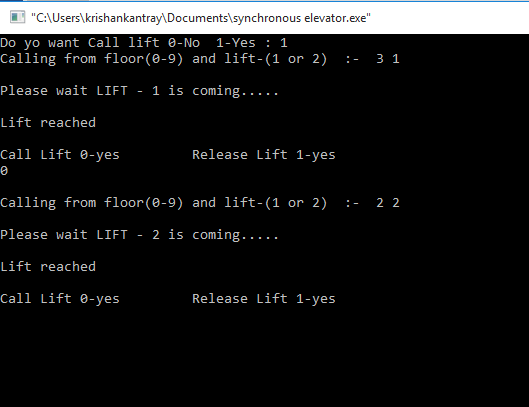
}

}

}

**Partial Implementation:**

A sample implementaition of the algorithm is mentioned below ( Command line output) :



**References:**

🡪 Introduction to Algorithms is a book by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein

🡪 Wikipedia.org

🡪 geeksforgeeks.org

🡪 https://msdn.microsoft.com

**Approved By**

**Project Guide PBL faculty coordinator**

No. of times you visited your guide:\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Approved by guide:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_