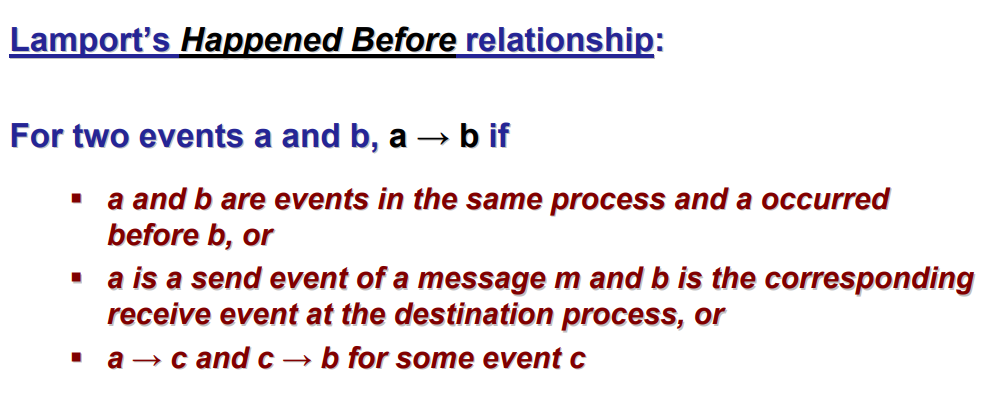
Distributed Systems (CS304)

Assignment - 8

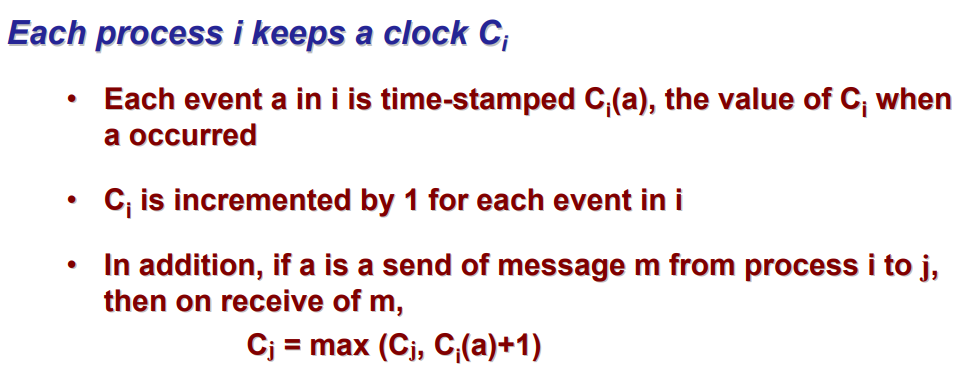
**U19CS012**

Q.) Implement **Lamport Logical Clock** Algorithm.

“Happened Before” Relation / Casual Ordering



**Algorithm**



**Code**

*// Implementation of Lamport Clock Algorithm [U19CS012]*

*#include* <bits/stdc++.h>

using namespace std;

typedef vector<int> vi;

typedef pair<int, int> pi;

*// Data Structure to Store all the Communication Lines {pid1,eid1} -> {pid2,eid2}*

typedef pair<pi, pi> ppipi;

*// Custom Comparator Function to Sort all the Communication Lines [Vector Clock]*

bool my\_sort(ppipi a, ppipi b)

{

*// Sort by Receiving Node {pid,eid} in ascending Order & Sending Node in descending Order*

*return* ((a.second.second < b.second.second) && (a.second.second < b.first.second) && (a.second.first < b.second.second) && (a.second.first < b.first.second) && (a.first.first > b.first.first));

}

int main()

{

*// freopen("input1.txt", "r", stdin);*

*// freopen("input2.txt", "r", stdin);*

*// freopen("input3.txt", "r", stdin);*

*// There are n Process namely P1, P2, P3,...PN*

    int n;

    cout << "\nEnter the Number of Processes : ";

    cin >> n;

    cout << '\n';

*// Store the Number of Events in Each Process*

    vi events(n, 0);

    int max\_events = 0;

*for* (int pid = 1; pid <= n; pid++)

    {

        int evnts;

        cout << "Enter the Number of Events in Process " << pid << " : ";

        cin >> evnts;

*// Update the Maximum Number of Events*

*if* (evnts > max\_events)

            max\_events = evnts;

        events[pid - 1] = evnts;

    }

    cout << '\n';

*// Input the Communication Lines*

    int comm\_lines;

    cout << "Enter the Number of Communication Lines : ";

    cin >> comm\_lines;

    cout << '\n';

*// Data Structure to Store the Communication Lines*

    vector<ppipi> lines;

*for* (int c = 0; c < comm\_lines; c++)

    {

        cout << "Communication Line Number " << c + 1 << " : \n";

*// For Each Communication Line {pid1,i1} -> {pid2,i2}*

        int pid1, eid1, pid2, eid2;

        cout << "Enter the Co-Ordinates {process\_id,event\_id} of Sending Node : ";

        cin >> pid1 >> eid1;

*// Valid Input Checks 1*

        assert(pid1 >= 1 && pid1 <= n);

        assert(eid1 >= 1 && eid1 <= events[pid1 - 1]);

        cout << "Enter the Co-Ordinates {process\_id,event\_id} of Receiving Node : ";

        cin >> pid2 >> eid2;

*// Valid Input Checks 2*

        assert(pid2 >= 1 && pid2 <= n);

        assert(eid2 >= 1 && eid2 <= events[pid2 - 1]);

        lines.push\_back({{pid1, eid1}, {pid2, eid2}});

        cout << '\n';

    }

    sort(lines.begin(), lines.end(), my\_sort);

*// Sorted Communication Lines*

    cout << "Communication Lines after Custom Sorting : \n";

*for* (int i = 0; i < lines.size(); i++)

    {

        cout << lines[i].first.first << " " << lines[i].first.second << " -> " << lines[i].second.first << " " << lines[i].second.second << "\n";

    }

    cout << '\n';

*// Lamport Clocks*

    vector<vector<vi>> vec(n, vector<vi>(max\_events, vi(1, 0)));

*// ? Intialize all the Lamport Clock(s) with Rule 1*

*for* (int i = 0; i < n; i++)

*for* (int j = 0; j < events[i]; j++)

            vec[i][j][0] = j + 1;

*// ? Implement Lamport Clock Algorithm {Rule 2}*

    int p1, e1, t1, p2, e2, t2;

*for* (int x = 0; x < comm\_lines; x++)

    {

*// Since Zero Based Indexing*

        p1 = lines[x].first.first - 1;

        e1 = lines[x].first.second - 1;

        p2 = lines[x].second.first - 1;

        e2 = lines[x].second.second - 1;

        vec[p2][e2][0] = max(vec[p2][e2][0], vec[p1][e1][0] + 1);

*// ! [IMP] Update the Following Lines after 'e2' Event, So it Reflects in Other Process as well*

*for* (int k = e2 + 1; k < events[p2]; k++)

        {

            vec[p2][k][0] = vec[p2][k - 1][0] + 1;

        }

    }

*// Print the Time Stamps of All the Processes*

*for* (int pid = 0; pid < n; pid++)

    {

        cout << "Process " << pid + 1 << " : ";

*// Intial Vector Clock*

        cout << "(" << 0 << ") : ";

*// Remaining Vector Clocks of Process 'pid'*

*for* (int eid = 0; eid < events[pid]; eid++)

        {

            cout << "(" << vec[pid][eid][0] << ") ";

        }

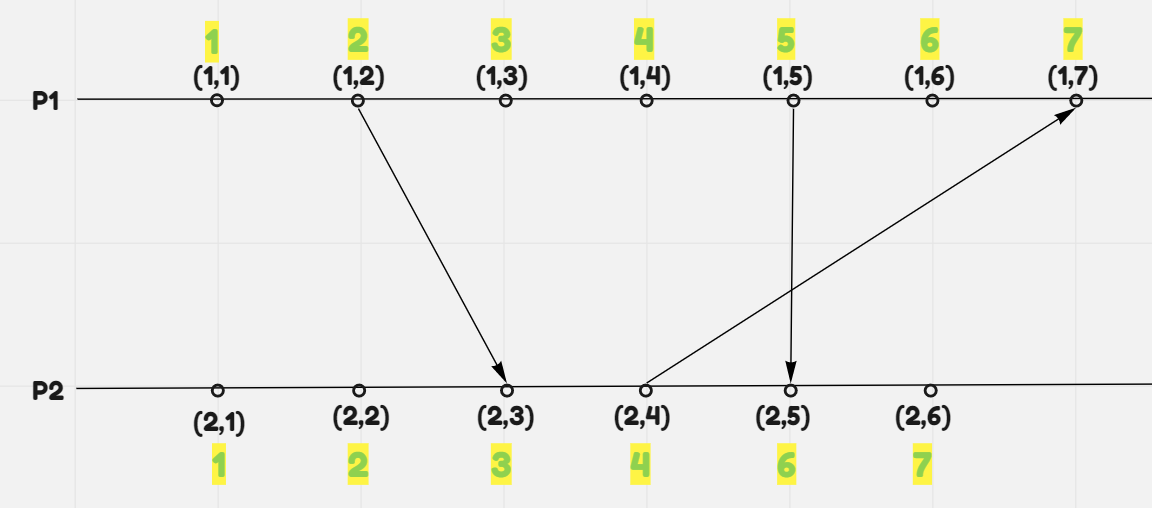
        cout << '\n';

    }

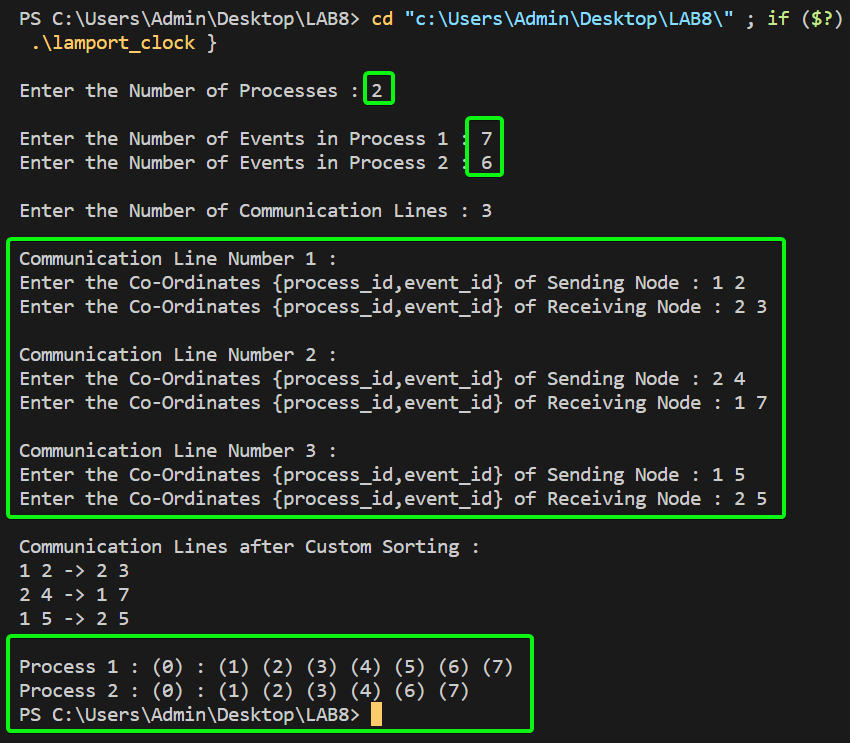
*return* 0;

}

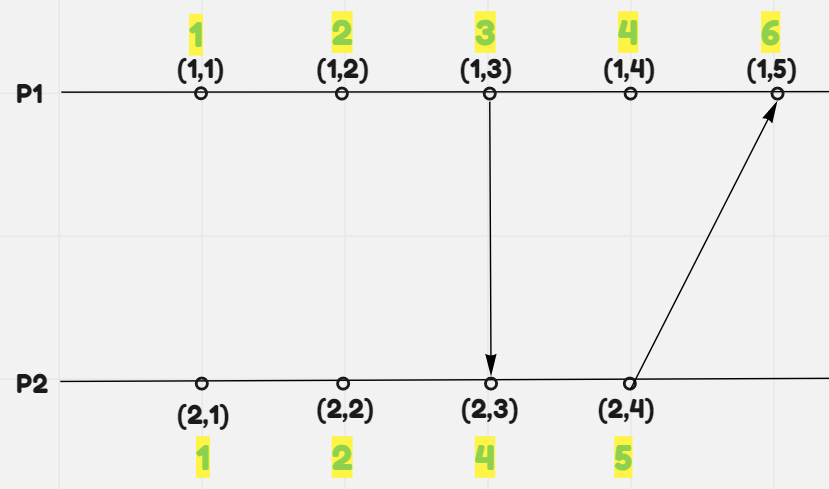
**Input (Basic)**



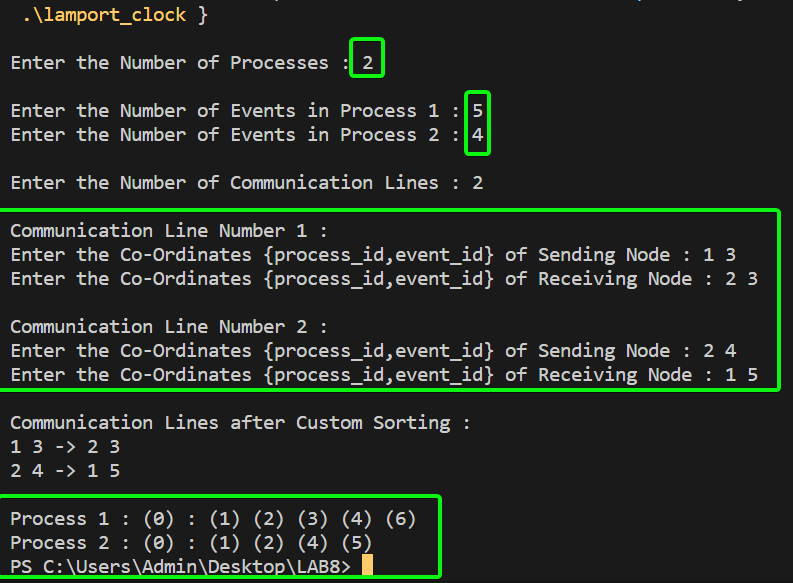
**Output**



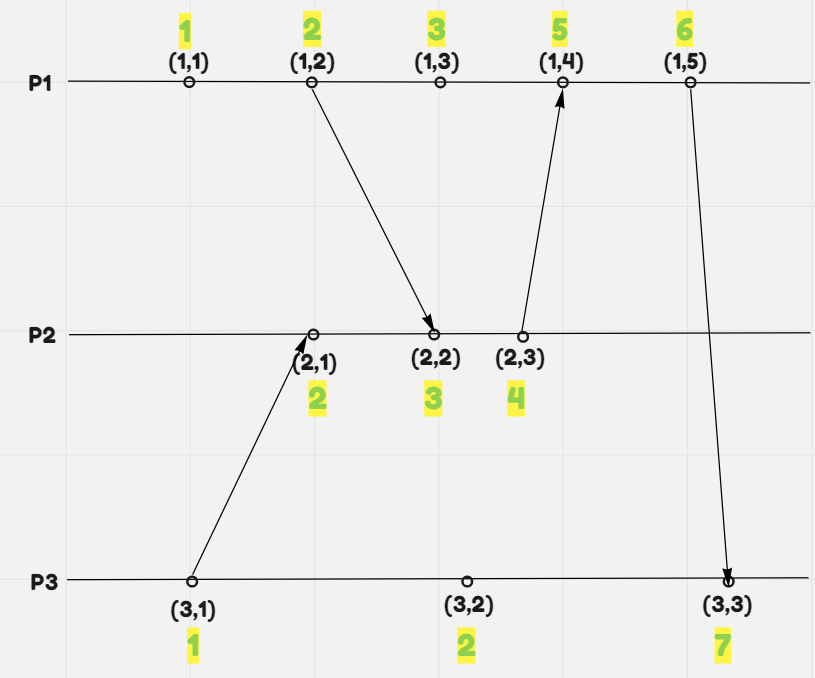
**Input (Medium)**



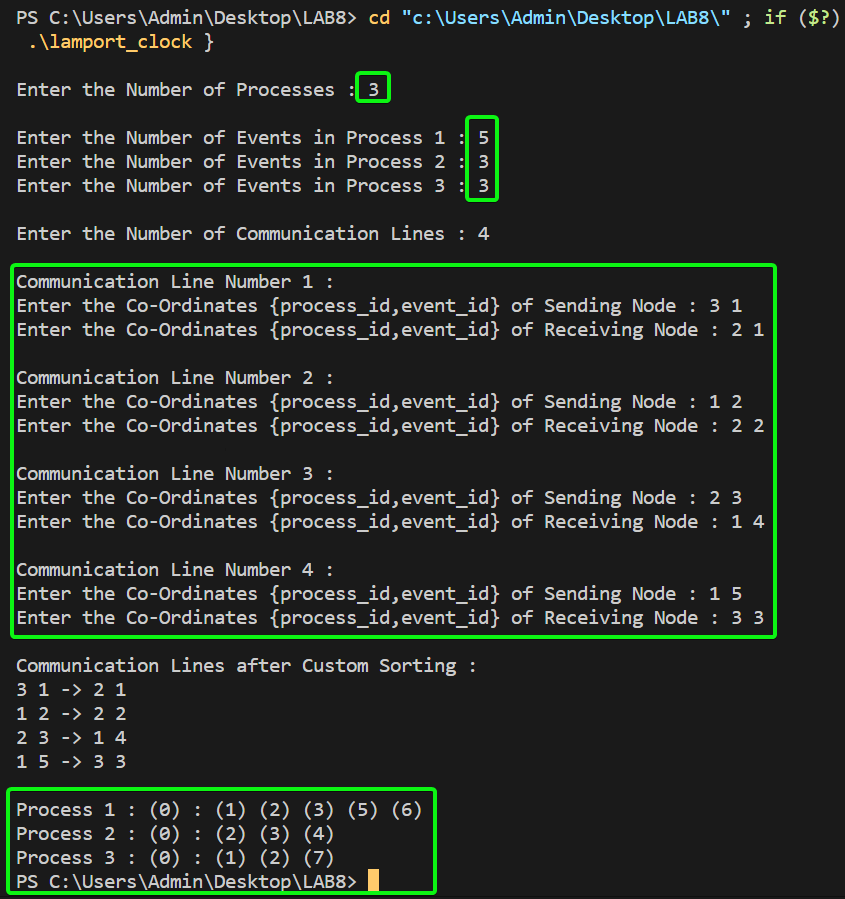
**Output**



**Input (Hard)**



**Output**



**SUBMITTED BY**: U19CS012

BHAGYA VINOD RANA