JATIYA KABI KAZI NAZRUL ISLAM UNIVERSITY

TRISHAL, MYMENSINGH

LAB REPORT

Course Name: VLSI Design Lab

Course Code: CSE-452

Submitted To

Dr. Md. Selim Al Mamun

Associate Professor

Dept. of CSE, JKKNIU

Submitted By

Deena Faria

Roll: 17102005

Reg.: 5682

Session: 2016-17

Submission Date: 11.06.2021

Experiment No.05

Experiment Name: Write a program to implement Critical Path Algorithm that can find a critical path in a graph.

Objective: Learn about critical path algorithm and to find critical path in a graph.

Theory:

Critical path algorithm

- The algorithm starts with finding the earliest possible start time for each node, going through the network.
- Then the latest possible start time for each node is found by going backward through the network.
- Nodes which have equal earliest possible start time and latest possible start time are on the critical path.

Program Code:

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

using namespace std;


struct activity {

    string name;

    int duration;

    int es, ef, ls, lf, st;  // es : earliest start time , ef : earliest finish time

                                                    // ls : latest start time ,  lf : latest finish time

                                                    // st : slack time

};
```

```
// returns vector of n numbers for input
vector<int> ReadNumbers()
{
   vector<int> numbers ;
   do
   {
      int input ;
      if (cin >> input )
         numbers.push_back(input) ;
   }while ( cin && cin.peek() != '\n' );

   return numbers ;
}

// utility for topological sorting of activity graph
void topologicalSortUtil(int v, vector<bool> &visited,  stack<int> &Stack, vector< vector<int> > &adj)
{
   visited[v] = true;
   vector<int>::iterator i;
   for (i = adj[v].begin(); i != adj[v].end(); ++i)
      if (!visited[*i])
         topologicalSortUtil(*i, visited, Stack, adj);
```

```
    Stack.push(v);
}


int main() {
    cout<<"############### Critical Path Algorithm #################\n\n";
    cout<<"Enter the number of tasks : ";
    //freopen ("input.txt","r",stdin);//input from file

    int i,n_tasks,top,j;
    cin>>n_tasks; // the number of tasks

    struct activity nodes[n_tasks+2]; // number of activities here 0th activity is the start
                                      // and the (n+1)th activity refers finish both having duration 0
    nodes[0].name = "Start";
    nodes[0].duration = 0;
    nodes[n_tasks+1].name = "Finish";
    nodes[n_tasks+1].duration = 0;

    // input of all the tasks
```

```
    for(i = 1 ; i <= n_tasks; i++) {
            cout<<"\n\nEnter task #"<<i<<" : ";
            cin>>nodes[i].name;
            cout<<"Enter duration for "<<i<<" : ";
            cin>>nodes[i].duration;

    }
   //print
    cout<<"\n\n\t\tTasks entered :\n";
    for(i = 0 ; i <= n_tasks+1; i++) {
            cout<<"\t\t"<<i<<".                                    "<<nodes[i].name<<"
"<<nodes[i].duration<<endl;
    }


    vector< vector<int> > adj;  // adj represents sucessor list
    vector< vector<int> > pred; // pred reperesents predecessor list

    // initialization of both lists with empty vectors
    for(i = 0 ; i <= n_tasks; i++) {
            vector<int> temp;
            adj.push_back(temp);
            pred.push_back(temp);
```

```
}

// initialization of successor list based on user input
for(i = 0 ; i <= n_tasks; i++) {
        cout<<"\n\nEnter successors for task "<<nodes[i].name<<" : ";
        vector<int> temp = ReadNumbers();
        if(temp.size()==0){
                adj[i].push_back(n_tasks);
                pred[n_tasks].push_back(i);

        }
        else {
                for(int j=0; j<temp.size(); j++)
                        adj[i].push_back(temp[j]);
                for(int j=0;j < temp.size(); j++)
                        pred[temp[j]].push_back(i);

        }

}
// calculating earliest start and finish times for each task
// topological sort of task is required here
```

```
stack<int> Stack;
vector<bool> visit(n_tasks+2, false);
topologicalSortUtil(0,visit, Stack, adj);

nodes[0].es = 0;
nodes[0].ef = 0;
Stack.pop();

while(!Stack.empty()) {
        top = Stack.top();
        int max_f = -1;
        for(i = 0; i < pred[top].size(); i++) {
                if(max_f < nodes[pred[top][i]].ef) {
                        max_f = nodes[pred[top][i]].ef;
                }
        }
        nodes[top].es = max_f;
        nodes[top].ef = max_f + nodes[top].duration;
        Stack.pop();
}

// calculating latest start and finish time for each task
```

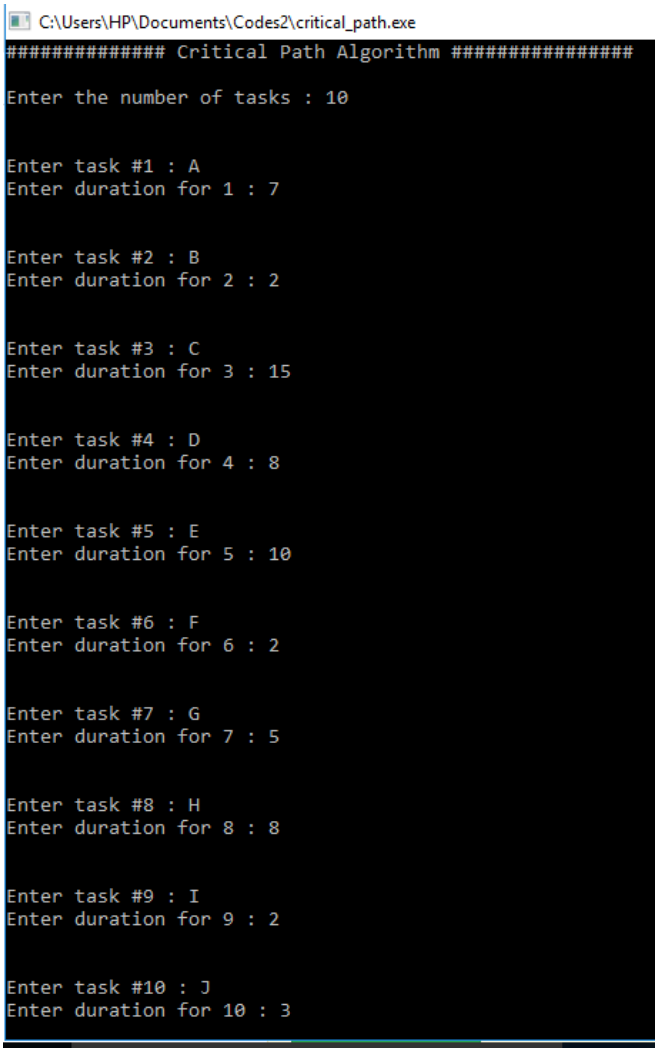
```
stack<int> Stack2;
vector<bool> visit2(n_tasks+2, false);
topologicalSortUtil(n_tasks+1, visit2, Stack2, pred);

nodes[n_tasks+1].ls = nodes[n_tasks+1].es;
nodes[n_tasks+1].lf = nodes[n_tasks+1].ef;
Stack2.pop();
while(!Stack2.empty()) {
        top = Stack2.top();
        int min_s = 99999;
        for(i = 0; i < adj[top].size(); i++) {
                if(min_s > nodes[adj[top][i]].ls) {
                        min_s = nodes[adj[top][i]].ls;
                }
        }
        nodes[top].lf = min_s;
        nodes[top].ls = min_s - nodes[top].duration;
        Stack2.pop();
}


// display of results
cout<<"RESULTS : \n\n";
```

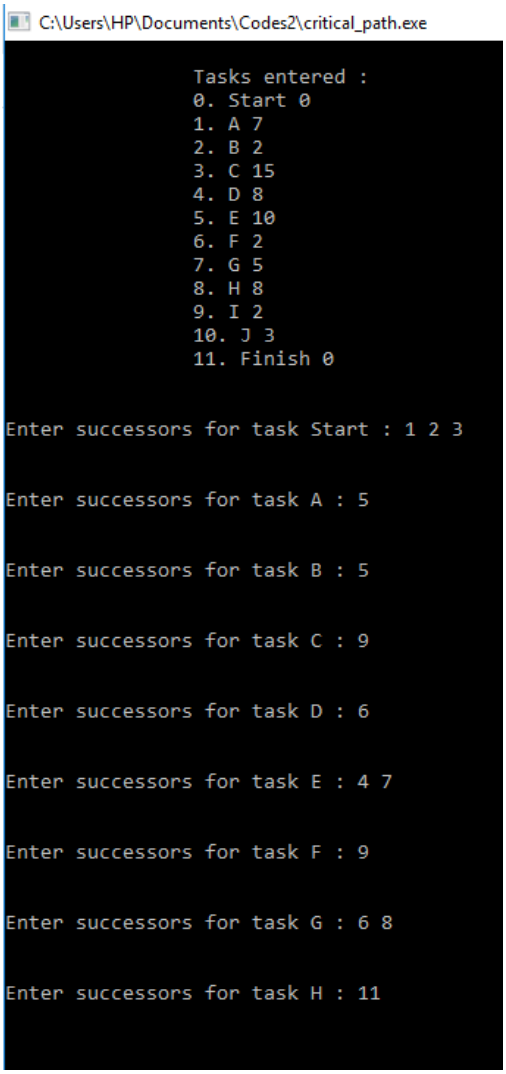
```
    cout<<"\t#\tTask\tDur.\tEs\tEf\tLs\tLf\tST\n\n";
    for(i = 0 ; i < n_tasks+2 ; i++) {
            nodes[i].st = nodes[i].ls - nodes[i].es;

    cout<<"\t"<<i<<"\t"<<nodes[i].name<<"\t"<<nodes[i].duration<<"\t"<<nod
es[i].es<<"\t"<<nodes[i].ef<<"\t"<<nodes[i].ls<<"\t"<<nodes[i].lf<<"\t"<<node
s[i].st<<"\n\n";
    }


  //simple BFS can be done to find critical path


    queue<int> q3;
    vector<int> visited3(n_tasks+2,0);


    vector<string> critical_path(n_tasks+2);///for path
    vector<string>::iterator it;


    q3.push(0);
    while(!q3.empty()) {
            top = q3.front();
            q3.pop();
            if(nodes[top].es == nodes[top].ls) {
         it = find (critical_path.begin(), critical_path.end(),nodes[top].name);
```

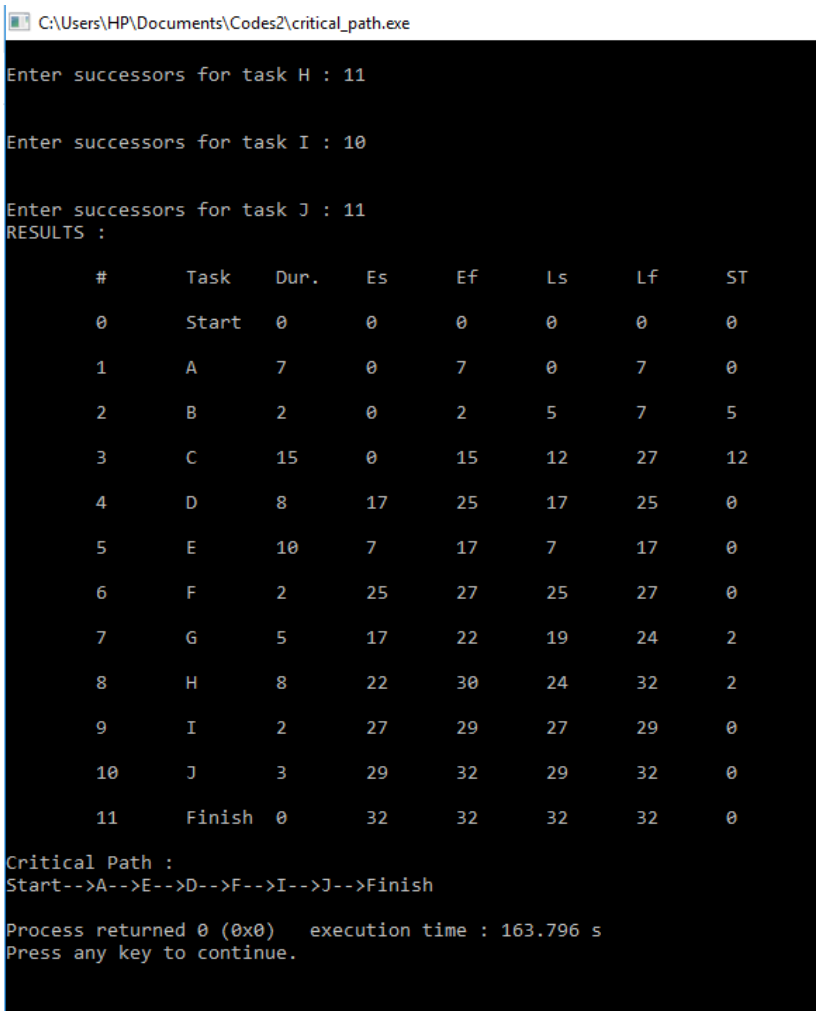
```
        if (it != critical_path.end())
          critical_path.erase(it);
                    critical_path.push_back(nodes[top].name);//store path here
          }
          for(i = 0 ; i < adj[top].size(); i++) {
                    if(visited3[adj[top][i]] == 0){
                              q3.push(adj[top][i]);
                    }
          }
}


cout<<"Critical Path : \n";
int len = critical_path.size();
for(i =0 ; i < len  ; i++) {
    if(critical_path[i]!=""){
         cout <<critical_path[i];
         if(i < len-1)
            cout  <<"-->";
    }
}

cout<<endl;
```

```
    return 0;


}
```

Output:







Discussion: We have successfully implemented critical path algorithm that can find a critical path in a graph.