DESIGN AND ANALYSIS OF ALGORITHM MINI-PROJECT REPORT

INTERNET NETWORKING USING TOPOLOGICAL ALGORITHIM

TEAM MEMBERS:

VUPPALA KAMALESH KUMAR(RA2011026010198)

S. UDAY(RA2011026010215)

G.SUBRAHMANYAM(RA2011026010188)

CONTENT:

CONTRIBUTION TABLE:

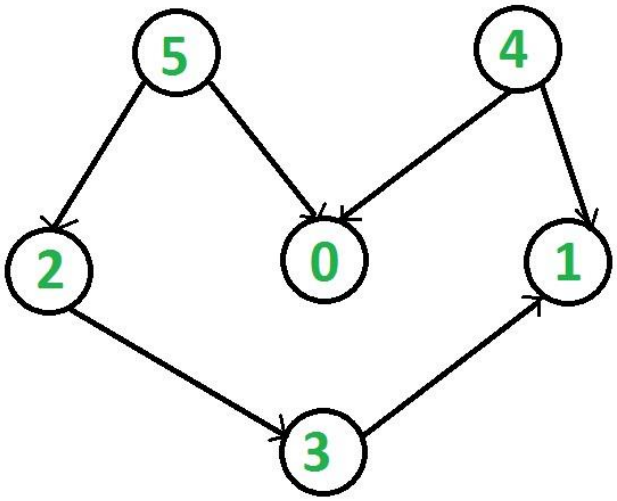
SLNO	Name	REG NO	Contribution
1	Vuppala kamalesh kumar	RA2011026010198	Intro, Problem statement
2	S.Uday	RA2011026010215	Problem analysis
3	G.Subrahymanyam	RA2011026010188	Complexity analysis

PROBLEM DEFINATION:

- The topological sort algorithm takes a directed graph and returns an array of the nodes where each node appears *before* all the nodes it points to
- The ordering of the nodes in the array is called a *topological ordering.*

PROBLEM EXPLINATION:

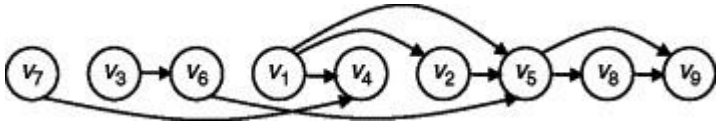
- By using topological algorithm we can know about internet network where it is high and where it is low.
- By this algorithm we can assume the graph we can rectify the problem where the network is slow.
- We implement a code by using C++ that helps in getting the graph that in which cities or area the network is lag.
- This the useful technique that we can find easily. That means by using this graph system we can also know the how much frequency is getting in that area or city.



WHAT IS TOPOLOGICAL ALGORITHM: A *topological .sort* is a linear ordering of vertices in a directed acyclic graph (DAG). Given a DAG $G = (V, E)$, a topological sort algorithm returns a sequence of vertices in which the vertices never come before their predecessors on any paths. In other words, if $(u, v) \in E$, v never appears before u in the sequence. A topological sort of a graph can be represented as a horizontal line of ordered vertices, such that all edges point only to the right.

ADVANTAGES OF TOPOLOGICAL ALGORITHM:

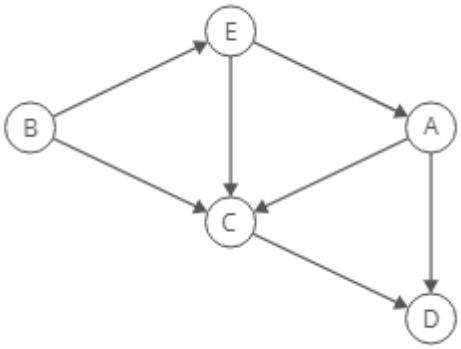
- Topological Sorting is mostly used to schedule jobs based on their dependencies. Instruction scheduling, ordering formula cell evaluation when recomputing formula values in spreadsheets, logic synthesis, determining the order of compilation tasks to perform in make files, data serialization, and resolving symbol dependencies in linker are all examples of applications of this type in computer science.



DESIGN TECHNIQUE USED:

- Topological sorting for Directed cyclic Graph (DAG) is a linear ordering of vertices such that for every directed edge u V, vertex u comes before v in the ordering. Topological Sorting for a graph is not possible if the graph is not a DAG.
- For example, a topological sorting of the following graph is "5 4 2 3 1 0". There can be more than one topological sorting for a graph. For example, another topological sorting of the following graph is "4 5 2 31 0". The first vertex in topological sorting is always a vertex with in- degree as 0 (a vertex with no incoming edges) **ALGORITHM OF THE PROBLEM:**

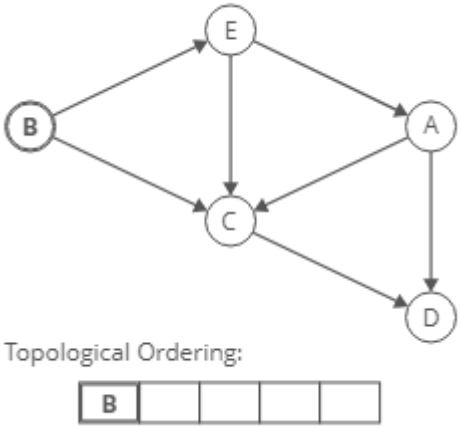
Step-1:Create the graph by calling addedge(a,b)



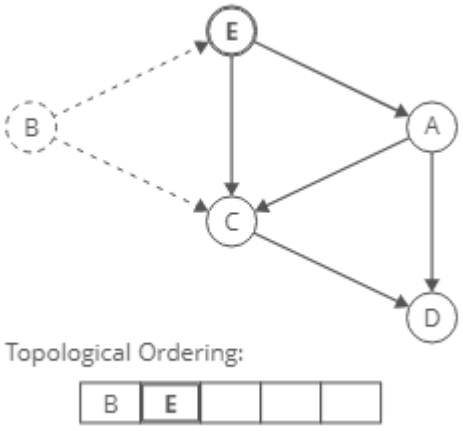
Step-2: call the topologicalsort()

Step-3: def topological sort util(int v, bool visited[],stack<int> &stack)

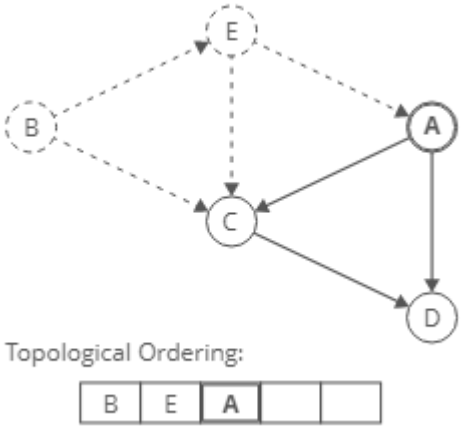
Step-4: atleast after return from the utility function,print contents of stack.\

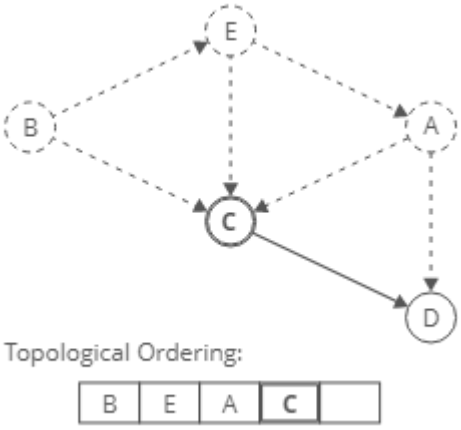


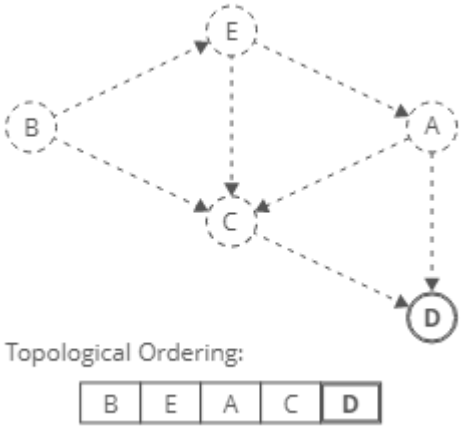
STEP-5: Once a node is added to the topological ordering, we can take the node, and its outgoing edges, out of the graph.



Repeat…







PSEUDO CODE:

//input: A DAG

//output: A list of the vertices of G in topological order

S = Stack()

L = List 0

For each vertex G:

If vertex has no incident edges:

S.push(vertex)

While S is not empty:

V = S.pop()

L.append(v)

For each outgoing edge e from v:

W=e.desitination

Delete e

If w has no incident edges:

s.push()

Return L

IMPLEMENTA

TION OF

CODE:

```
    #include <iostream>
#include          <list>
#include       <stack>
using namespace std;

// Class to represent a graph class
Graph {
    // No. of vertices' int
    V;

    // Pointer to an array containing adjacency listsList list<int>*
    adj;
// A function used by topologicalSort void
    topologicalSortUtil(int v, bool visited[],
    stack<int>& Stack);

public:
    // Constructor Graph(int
    V);
```

```
// function to add an edge to graph void
    addEdge(int v, int w);

    // prints a Topological Sort of
/ the complete graph void
    topologicalSort();
};

Graph::Graph(int V)
{
    this->V = V; adj =
    new list<int>[V];
}

void Graph::addEdge(int v, int w)
{
// Add w to v's list.
    adj[v].push_back(w);
}

// A recursive function used by topologicalSort void
Graph::topologicalSortUtil(int v, bool visited[],
stack<int>& Stack)
{
// Push current vertex to stack
    // which stores result
    Stack.push(v);
}
// Mark all the vertices as not visited
    bool* visited = new bool[V]; for
    (int i = 0; i < V; i++)
        visited[i] = false;
// Call the recursive helper function
```

```
    // to store Topological
    // Sort starting from all
    // vertices one by one
for (int i = 0; i < V; i++) if
(visited[i] == false)
            topologicalSortUtil(i, visited, Stack);
// Print contents of stack while
    (Stack.empty() == false) {
cout << Stack.top() << " "; Stack.pop();
    }
}


// Driver Code int
main()
{
    // Create a graph given in the above diagram
    Graph g(6);
    g.addEdge(5, 2);
    g.addEdge(5, 0);
    g.addEdge(4, 0);
    g.addEdge(4, 1);
    g.addEdge(2, 3);
    g.addEdge(3, 1); cout << "Following is a Topological Sort of the given
"
            "graph \n";

    // Function Call
    g.topologicalSort(); return
    0;
}
```

INPUT:

//entering the number of vertices

6

//entering the number of edges

5

//entering edges one by one in format repeatedly

5 2
5 0
4 0
4 1
2 3
3 1

OUTPUT: The output of the above code is

Edges are
5
4
2
3
1
0

TIME COMPLEXITY :

- Uses DFS
- Topological sort(G)
- call DFS(G) to compute finishing times f(v) for each v output vertices in decreasing order of finishing time

- Time complexity is O((V)+(E)).

REASON:

The time complexity of topological sort using Kahn's algorithm is O(V+E), where V = Vertices, E = Edges. To find out the time complexity of the algorithm, let's try to find the time complexity of each of the steps: To determine the in-degree of each node, we will have to iterate through all the edges of the graph**.**

SPACE COMPLEXITY:

Auxiliary space required is O(V). We have to create one array to store the in-degrees of all the nodes. This will require O(V) space. We have to store the nodes with in-degree = 0 in a data structure (stack or queue) when we remove them from the graph.

CONCLUSION:

- So, here we done the internet networking using topological algorithm.
- It is used for partial defined orders. In recent years, the algorithm was developed for parallel computation. In general use, this algorithm is used in cases for which a graphic model is available for people to understand bigger problems.

REFERENCES:

- GEEKS FOR GEEKS
- ONLINE GDB
- FREECODE CAMP • INTERVIEW CAKE
- JAVA POINT.