CSCI 2270 – Data Structures

Recitation 11, November 2018

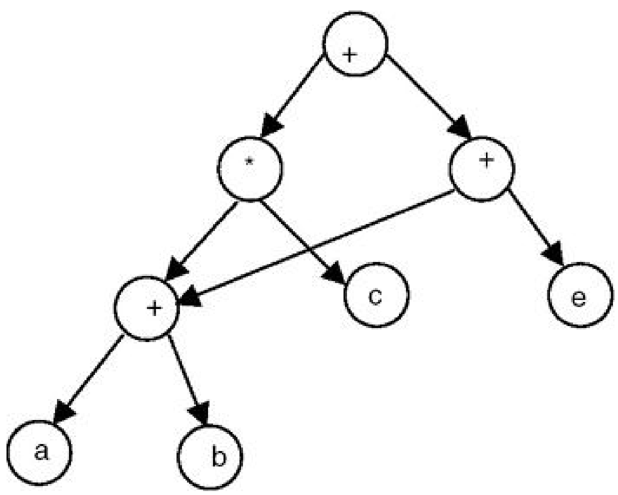
Advanced Problems

Objectives

1. Topological Sorting
2. Zig-Zag Tree traversal

Directed Acyclic Graph (DAG)

A graph is a series of vertices connected by edges. In a directed graph, the edges are connected so that each edge only goes one way. A directed acyclic graph means that the graph is not cyclic, or that it is impossible to start at one point in the graph and traverse the entire graph. For example - consider the graph below.



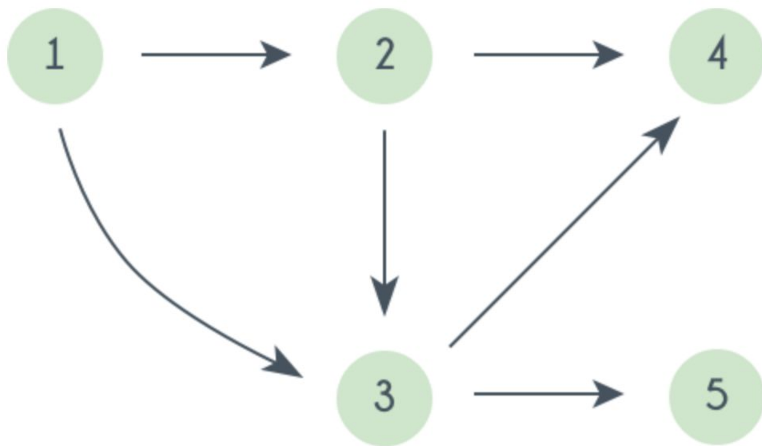
Topological Sort

Topological sorting of vertices of a Directed Acyclic Graph is an ordering of the vertices v1,v2,...vn in such a way, that if there is an edge directed towards vertex vj from vertex vi, then vi comes before vj. For example consider the graph given below:

CSCI 2270 – Data Structures

Recitation 11, November 2018

Advanced Problems



A topological sorting of this graph is: 1 2 3 4 5

There are multiple topological sorting possible for a graph. For the graph given above one another topological sorting is: 1 2 3 5 4

Two algorithmic questions:

1. Given an (ordered) list of all vertices in the graph, is it a topological ordering?
2. **Given a graph, produce a topological ordering.**

For the second question, to produce a topological ordering, our first question can be at which vertex we should start?

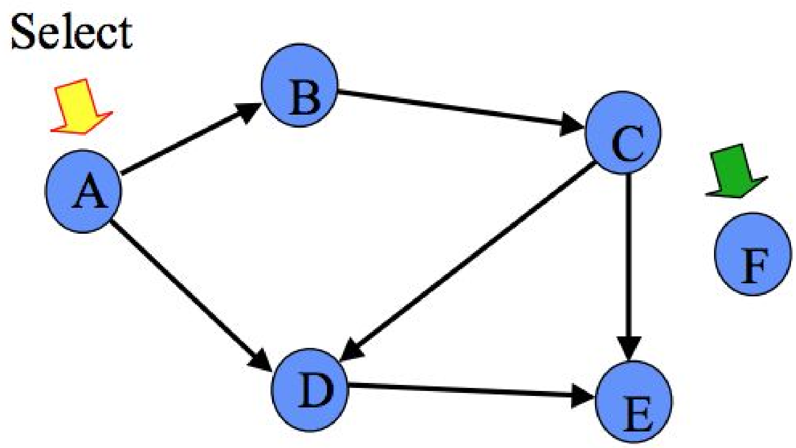
Step 1: Identify vertices that have no incoming edge. In a DAG, vertices with no incoming edges are called sources.
Every DAG has a or at least one source (you can easily prove this!).

Select one such vertex/source.

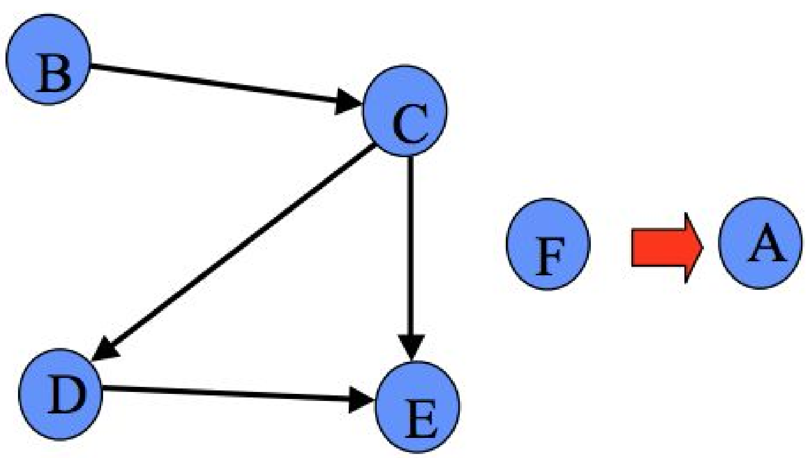
CSCI 2270 – Data Structures

Recitation 11, November 2018

Advanced Problems

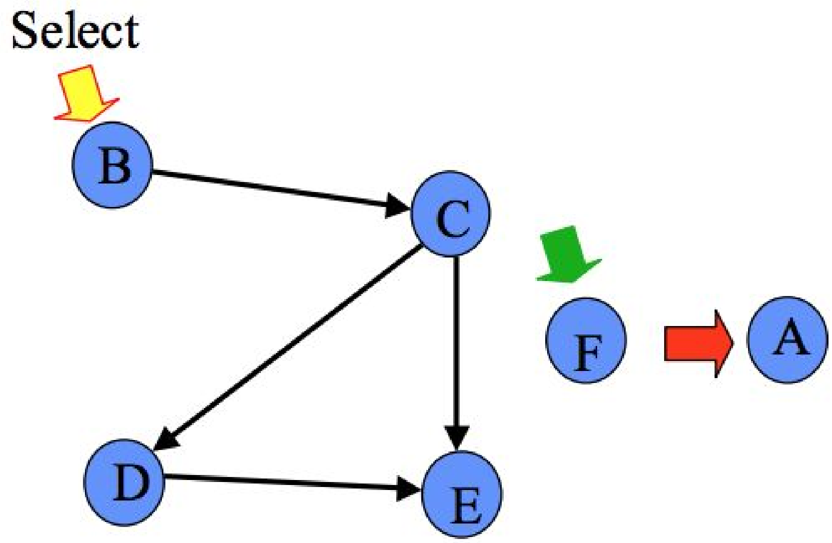


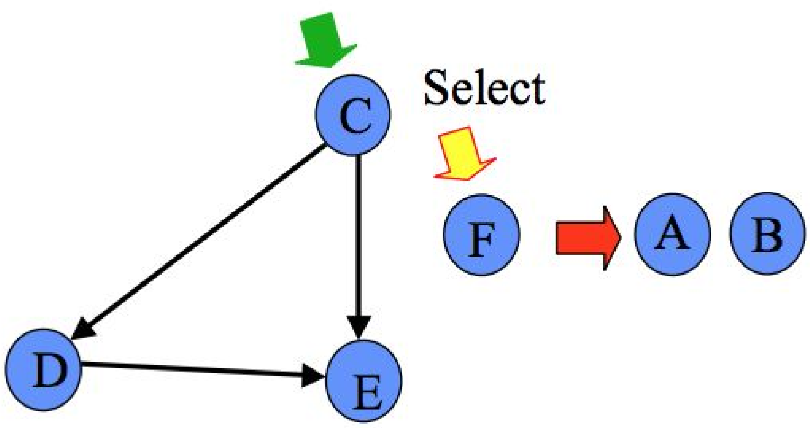
Step 2: Delete this vertex of in-degree 0 and all its outgoing edges from the graph. Place it in the output.
In Degree: This is applicable only for directed graph. This represents the number of edges incoming to a vertex.



Repeat Step 1 and Step 2 until graph is empty.

Advanced Problems



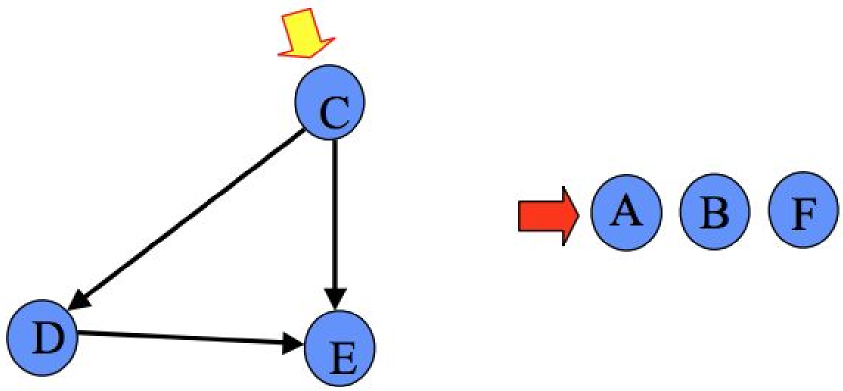


CSCI 2270 – Data Structures

Recitation 11, November 2018

Advanced Problems

Select



Final Result:

We'll maintain an array T that will denote our topological sorting. So, let's say for a graph having N vertices, we have an array in_degree[] of size N whose ith element tells the number of vertices which are not already inserted in T and there is an edge from them incident on vertex numbered i.

The algorithm using a BFS traversal is given below:

```
topological_sort(N, adj[N][N])
        T = []
        visited = []
        in_degree = []
        for i = 0 to N
                in_degree[i] = visited[i] = 0

        for i = 0 to N
                for j = 0 to N
                        if adj[i][j] is TRUE
```

```
                                in_degree[j] = in_degree[j] + 1

        for i = 0 to N
                if in_degree[i] is 0
                        enqueue(Queue, i)
                        visited[i] = TRUE

        while Queue is not Empty
                vertex = get_front(Queue)
                dequeue(Queue)
                T.append(vertex)
                for j = 0 to N
                        if adj[vertex][j] is TRUE and visited[j] is FALSE
                                in_degree[j] = in_degree[j] - 1
                                if in_degree[j] is 0
                                        enqueue(Queue, j)
                                        visited[j] = TRUE
        return T
```

Applications of Topological Sort

1) Course Scheduling Problem
2) Finding total number of paths from a source to destination

Zig Zag tree Traversal

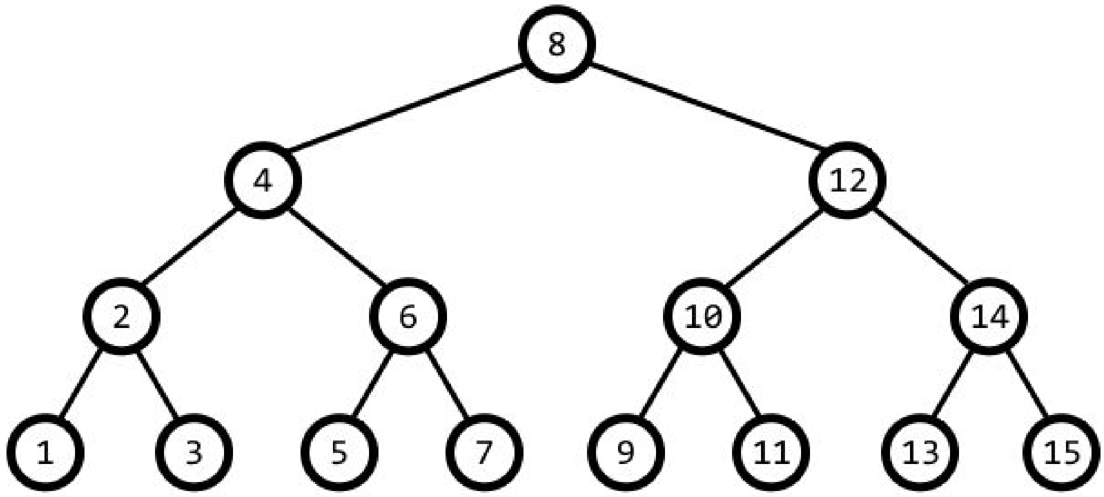
Given a complete binary tree. Print the elements of the tree in a zig-zag fashion.
Definition of Zig-Zag

1. In level order we printed out elements level by level.
2. We printed the elements left to right at each level.
3. But in Zig-Zag traversal, we alter this sequence at every level.
4. The odd indexed levels are printed in a left-right fashion.
5. The even indexed levels are printed in a right-left manner.

CSCI 2270 – Data Structures

Recitation 11, November 2018

Advanced Problems



Example:

Level Order - 8, 4, 12, 2, 6, 10, 14, 1, 3, 5, 7, 9, 11, 13, 15

Zig-Zag - 8, 12, 4, 2, 6, 10, 14, 15, 13, 11, 9, 7, 5, 3, 1

This problem can be solved using a stack and a queue. For even indexed levels, you need to print them left-right. So a normal level order traversal suffices in this case. However for odd indexed levels, you should do a level order traversal and put it in a stack. Then, when that level is done, pop each element from the stack and then print them in a reverse manner. We can insert special markers into the queue to indicate levels.
There are multiple ways to solve this problem. You are free to implement your own version.

Exercise

1. Download the **ZigZagMain.cpp** and **BinaryTree.hpp** file from moodle.
2. Complete the **ZigZagTraverse** method by using the logic mentioned above.