College of Computing and Information Technology

Lecturer: Dr. Nahla Belal

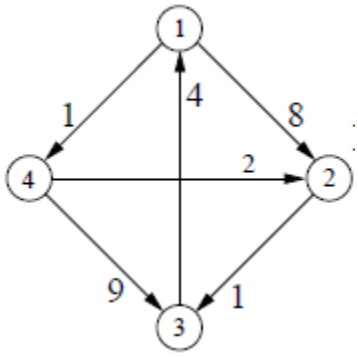
Course: Computing Algorithms (CS312)

TA: Eng. Ahmad Goudah

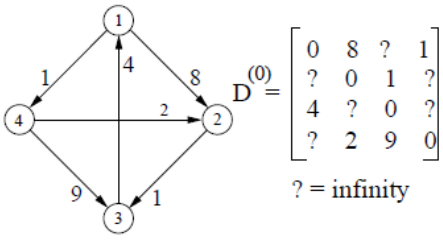
Sheet 7

Graph Theroy Problems:

1 – (Floyd-Warshall Algorithm) Suppose You have a weighted, directed graph and want to find the shortest path from u to v for all pairs of vertices (u, v). Weighted, directed graph is a collection vertices connected by weighted edges (where the weight is some real number). Determine the shortest path between all pairs of vertices in a graph using The Floyd-Warshall algorithm. What is the complexity of this algorithm?

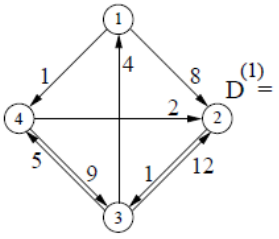


Sol:

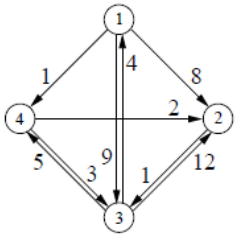


$D^{(0)} = \begin{bmatrix} 0 & 8 & ? & 1 \\ ? & 0 & 1 & ? \\ 4 & ? & 0 & ? \\ ? & 2 & 9 & 0 \end{bmatrix}$

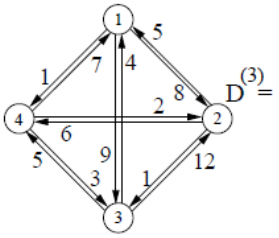
? = infinity



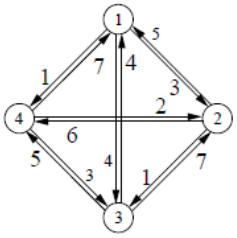
$D^{(1)} = \begin{bmatrix} 0 & 8 & ? & 1 \\ ? & 0 & 1 & ? \\ 4 & 12 & 0 & 5 \\ ? & 2 & 9 & 0 \end{bmatrix}$



$D^{(2)} = \begin{bmatrix} 0 & 8 & 9 & 1 \\ ? & 0 & 1 & ? \\ 4 & 12 & 0 & 5 \\ ? & 2 & 3 & 0 \end{bmatrix}$



$D^{(3)} = \begin{bmatrix} 0 & 8 & 9 & 1 \\ 5 & 0 & 1 & 6 \\ 4 & 12 & 0 & 5 \\ 7 & 2 & 3 & 0 \end{bmatrix}$



$D^{(4)} = \begin{bmatrix} 0 & 3 & 4 & 1 \\ 5 & 0 & 1 & 6 \\ 4 & 7 & 0 & 5 \\ 7 & 2 & 3 & 0 \end{bmatrix}$

```
Floyd_Warshall(int n, int W[1..n, 1..n]) {
    array d[1..n, 1..n]
    for i = 1 to n do {                              // initialize
        for j = 1 to n do {
            d[i,j] = W[i,j]
            pred[i,j] = null
        }
    }
    for k = 1 to n do                                // use intermediates {1..k}
        for i = 1 to n do                            // ...from i
            for j = 1 to n do                        // ...to j
                if (d[i,k] + d[k,j]) < d[i,j]) {
                    d[i,j] = d[i,k] + d[k,j]         // new shorter path length
                    pred[i,j] = k                    // new path is through k

                }
    return d                                         // matrix of final distances
}
```

Implementation:

```
int graph[128][128], n;                // a weighted graph and its size

void floydWarshall() {
    for( int k = 0; k < n; k++ )
    for( int i = 0; i < n; i++ )
    for( int j = 0; j < n; j++ )
        graph[i][j] = min( graph[i][j], graph[i][k] + graph[k][j] );
}

int main {
    // initialize the graph[][] adjacency matrix and n
    // graph[i][i] should be zero for all i.
    // graph[i][j] should be "infinity" if edge (i, j) does not exist
    // otherwise, graph[i][j] is the weight of the edge (i, j)

    floydWarshall();

    // now graph[i][j] is the length of the shortest path from i to j
}
```

Time complexity of the Floyd-Warshall algorithm is O(n3). The space used by the algorithm is O(n2).

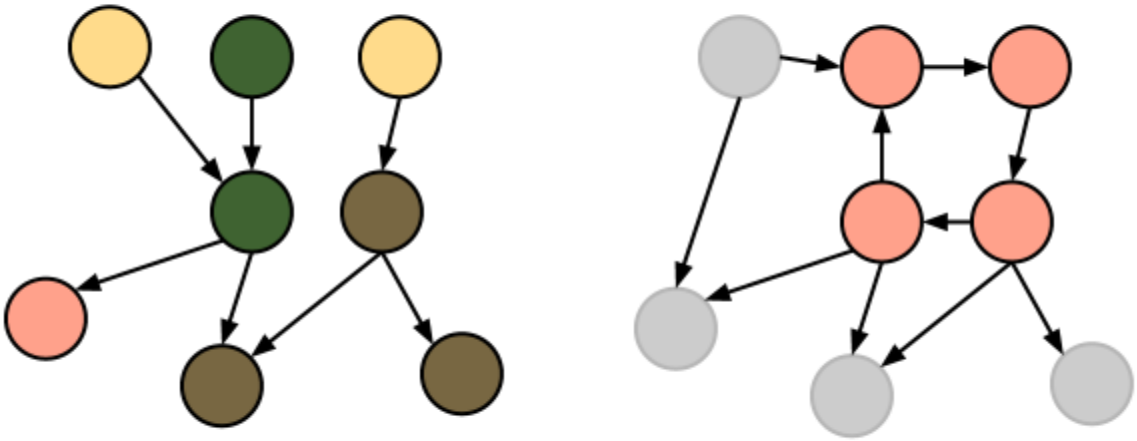
.

2 - Sort the execution of different tasks that depend on each other using **Topological Sort of a Graph.**

Let's assume we have a list of tasks to accomplish. Some of the tasks depend on others, so we must be very careful with the order of their execution. If the relationship between these tasks were simple enough we could represent them as a linked list, which would be great, and we would know the exact order of their execution. The problem is that sometimes the relations between the different tasks are more complex and some tasks depend on two or more other tasks, which in their turn depend on one or more tasks, etc.

Thus we can't model this problem using linked lists or trees. The only rational solution is to model the problem using a graph. What kind of graph do we need? Well, we definitely need a directed graph, to desribe the relations, and this graph shouldn't have cycles. So we need the so called directed acyclic graph (DAG).

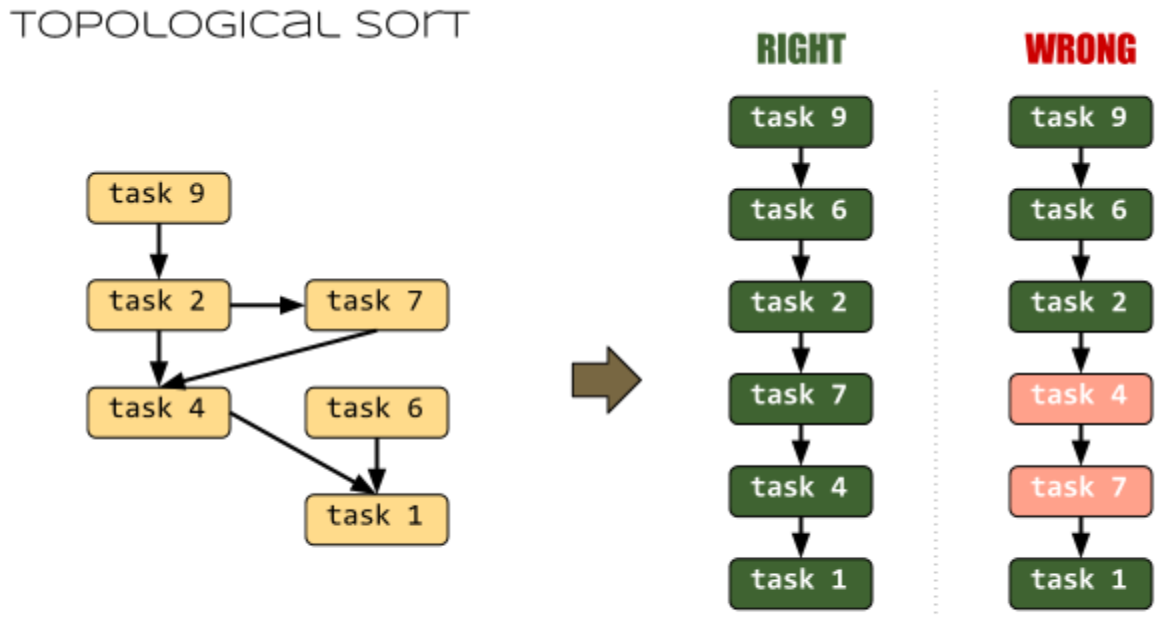
Directed Graph



Toplogical sort can be applied only to directed acyclic graph - (DAG).

In order to sort a graph using topological sort we need this graph to be acyclic and directed! Why we don't what a cycle in the graph? The answer of this question is simple and obvious. In case of cyclic graph, we wouldn't be able to determine the priority of task execution, thus we won't be able to sort the tasks properly.

Now the solution we want is to sort the vertices of the graph in some order so for each edge (u, v) u will precede v. Then we'll have a linear order of all tasks and by starting their execution we'll know that everything will be OK.



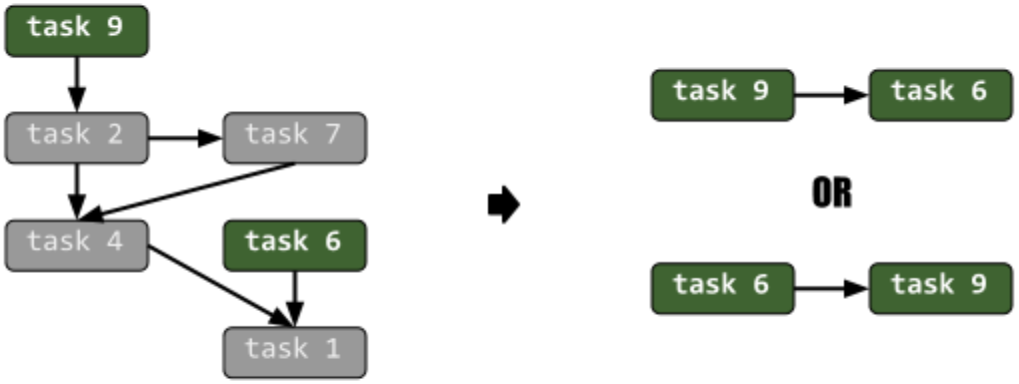
For each edge (u,v) u must be before v!

The output of topological sort should be a list of vertices!
This kind of sort is also known as “topological” sort (or topsort) and it is one of the very basic graph algorithms.

Overview

OK, so we have an acyclic directed graph, how do we proceed to get a linked list with all the vertices sorted? Since it’s an acyclic graph we know that there is at least one vertex without predecessor. Thus at first place, we can put all the vertices without predecessors into our linked list.

TOPOLOGICAL SORT



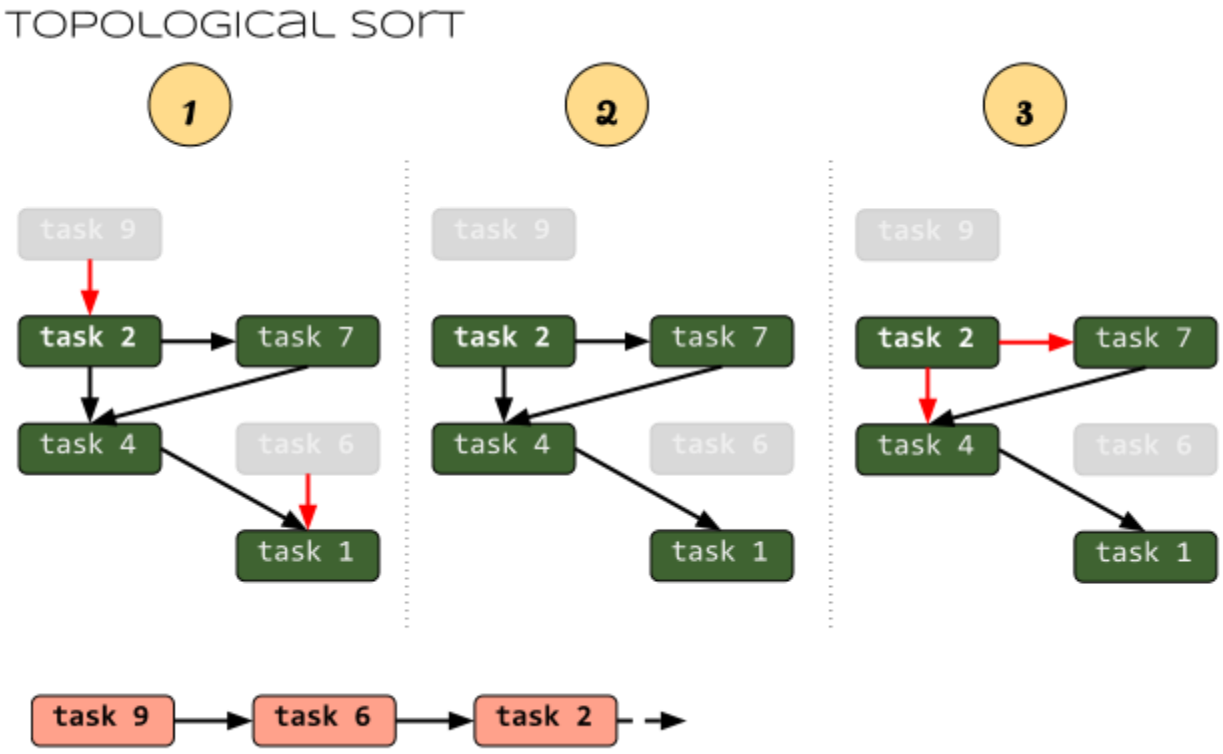
Collect all the vertices with no predecessors!

Initially we get only the vertices without a predecessor!
This approach answers the question – is there a possibility to have more than one valid topological sort of a graph? Indeed, the only thing we'd like to do is to put all the vertices in the correct order, but since there might be vertices with no predecessors any combination of them will be a valid topological sort for a graph.

As we can see from the picture above even for vertices with predecessors our topological sort can vary. Thus [9, 6, 2, 7, 4, 1] is a valid topological sorted graph, but [6, 9, 2, 7, 4, 1] is also a valid topological sort out of the same graph!

Now we can generalize the algorithm in some basic steps.

1. Make an empty list L and an empty list S;
2. Put all the vertices with no predecessors in L;
3. While L has items in it;
3.1. Pop an item from L – n, and push it to S;
3.2. For each vertex m adjacent to n;
3.2.1. Remove (n, m);
3.2.2. If m has no predecessors – push it to L;



The image above explains step 3.2. from the algorithm!

Code

Here's the very basic PHP implementation. As you can see the short implementation shows us how easy this algorithm is. However its importance to computer science and programming is enormous.

```
class G
{
    protected $_g = array(
        array(0, 1, 1, 0, 0, 0, 0),
        array(0, 0, 0, 1, 0, 0, 0),
        array(0, 0, 0, 0, 1, 0, 0),
        array(0, 0, 0, 0, 1, 0, 0),
        array(0, 0, 0, 0, 0, 0, 1),
        array(0, 0, 0, 0, 0, 0, 1),
        array(0, 0, 0, 0, 0, 0, 0),
    );
    protected $_list = array();
    protected $_ts   = array();
    protected $_len  = null;

    public function __construct()
    {
        $this->_len = count($this->_g);

        // finds the vertices with no predecessors
        $sum = 0;
        for ($i = 0; $i < $this->_len; $i++) {
            for ($j = 0; $j < $this->_len; $j++) {
                $sum += $this->_g[$j][$i];
            }

            if (!$sum) {
                // append to list
                array_push($this->_list, $i);
            }
            $sum = 0;
        }
    }

    public function topologicalSort()
    {
        while ($this->_list) {
            $t = array_shift($this->_list);
            array_push($this->_ts, $t);

            foreach ($this->_g[$t] as $key => $vertex) {
                if ($vertex == 1) {
                    $this->_g[$t][$key] = 0;

                    $sum = 0;
```

```
                    for ($i = 0; $i < $this->_len; $i++) {
                        $sum += $this->_g[$i][$key];
                    }

                    if (!$sum) {
                        array_push($this->_list, $key);
                    }
                }
                $sum = 0;
            }
        }

        print_r($this->_ts);
    }
}

$g = new G();
/*
Array
(
    [0] => 0
    [1] => 5
    [2] => 1
    [3] => 2
    [4] => 3
    [5] => 4
    [6] => 6
)*/
$g->topologicalSort();
```

Ref: http://www.stoimen.com/blog/2012/10/01/computer-algorithms-topological-sort-of-a-graph/