# Problem description

Given a query represented as a DAG where edges are precedence constraints and nodes units of work, write a program that accepts a request that executes the query, or nodes of the DAG without violating precedence constraints.

# Specification

A request is a DagRequest class that encloses the DAG serialized in XML:

/// <summary>

/// Class that encapsulates fields related to a DAG request.

/// </summary>

public class DagRequest

{

/// <summary>

/// DAG serialized in XML

/// </summary>

public string DagXml { get; }

}

A very simple example of a DAG XML serialization is:

<DAG>

<Nodes>

<Node Id="0" >

<dependencies>

<Node Id="1"/>

<Node Id="2"/>

</dependencies>

</Node>

<Node Id="1">

<dependencies/>

</Node>

<Node Id="2">

<dependencies/>

</Node>

</Nodes>

</DAG>

You can use any XML library to parse the XML document. It is not the intent of this exercise to implement an XML parser. In this example, the right order of execution would be that 1 and 2 complete before 0. 1 and 2 can be executed concurrently.

You do not need to implement a protocol of any sort to submit a request to the server, nor for the server to send the response back to the client. Instead, the primary focus is the entry point of your program that accepts and executes a DagRequest through the following interface:

/// <summary>

/// Interface for the execution of a DAG.

/// </summary>

public interface IDagExecutor

{

/// <summary>

/// Processes a Dag request and returns a response: success or failure.

/// </summary>

/// <param name="request">The request to be processed</param>

/// <returns>The response.</returns>

Task<RequestResponse> ProcessRequestAsync(DagRequest request);

}

While processing the request your program needs to de-serialize and execute the DAG without violating precedence constraints. Note that the program needs to have support for processing multiple concurrent requests.

The nodes of the DAG are executed to completion by an execution engine that implements the following interface:

/// <summary>

/// Execution engine responsible to execute the nodes in the graph.

/// </summary>

public interface IDagNodeExecutor

{

/// <summary>

/// Executes a node in the graph.

/// </summary>

/// <param name="unitOfExecution">The node to be executed.</param>

/// <returns>0 if success, < 0 otherwise.</returns>

public Task<int> ExecuteAsync(IDagNode unitOfExecution);

}

Your program can have a pool of M execution engines, where each can only accept the execution of one IDagNode at a time.

Requirements

* The program needs to be able to accept multiple concurrent requests.
* The program needs to validate the correctness of the input DAG. A DAG with islands is valid.
* The program holds a pool of M execution engines, each with capacity to execute only one node of the DAG at a time. M can be configurable.
* The execution engine returns failure one out of N executions. N is configurable by the server.
* Programming language: Language of your choice. If using C# then we strongly recommend the use of async task programming. Please provide a zip folder with the solution file and source.
* Custom unit tests with test cases.
* Brief document explaining what’s done and design choices.
* Product quality code is expected in your solution.

Expected time to completion

12-16 hours. You can take more time if needed – we would like you to focus more on production quality solution than solving it withing the expected completion time.