```
    public interface IState {
        bool IsGoal();
    }

    public interface IOperator {
        bool IsApplicable(IState s);
        IState Apply(IState s);
    }

    public interface Problem {
        IState StartState();
        List<IOperator> Operators();
        double Cost(IState state, IOperator opr);
        double Heuristic(IState state);
    }
```

```
class TNode
{
     IState state;
     TNode parent;
     IOperator creator;
     double cost;

     void Update(TNode newParent, IOperator creator, Problem p)
     {
     ...
     }
}
```

```
class Node {
    public IState state;
    public Node parent;
    public IOperator creator;
    public List<IOperator> unused;
    public Node( IState state, Node parent, IOperator creator, Problem p ) {
        this.parent = parent;
        this.state = state;
        this.creator = creator;
        unused = new List<IOperator>();
        foreach ( var o in p.Operators() )
            if ( o.IsApplicable(state) )
                unused.Add(o);
    }
}

class Algorithm
{
     private Problem problem;
     public Algorithm(Problem p)
     {
         this.problem = p;
     }

     private Node actualNode = null;

     public bool Run()
     {
         actualNode = new Node(problem.StartState(), null, null, problem);
         while (true)
         {
             if (actualNode == null)
                 return false;
             if (actualNode.state.IsGoal())
                 return true;
             if (actualNode.unused.Count > 0)
             {
                 IOperator op = actualNode.unused.ElementAt(0);
                 actualNode.unused.RemoveAt(0);
                 IState newState = op.Apply(actualNode.state);
                 actualNode = new Node(newState, actualNode, op, problem);
             }
             else
             {
                 actualNode = actualNode.parent;
             }
         }
     }
}
```