

# Search!

## EECS 4315

[www.cse.yorku.ca/course/4315/](http://www.cse.yorku.ca/course/4315/)



# Breadth First Search

```
import gov.nasa.jpf.search.Search;

public class BFSearch extends Search {
    public Search(Config config, VM vm) {
        super(config, vm);
    }
}
```

# The search Method

The method

```
public void search()
```

drives the search.

```
public RestorableVMState getRestorableVMState()
```

returns a **RestorableVMState** object which represents the current state.

```
public void restoreState(RestorableVMState)
```

restores the given state, that is, the current state becomes a previously visited state provided as an argument.

```
public boolean isNewState()
```

tests whether the current state has not been visited before.

```
public boolean isEndState()
```

tests whether the current state is a final state.

```
public boolean isIgnoredState()
```

tests whether the current state can be ignored.

## Problem

Implement the `search` method of the `BFSearch` class.

# The done Attribute

Other components of JPF can end a search by setting the attribute **done** of the class **Search** to true.

## Problem

Modify the **search** method of the **BFSearch** class to incorporate the **done** attribute.

# Request Backtrack

The class `Search` contains the method `supportBacktrack` which tests whether a search supports backtrack requests.

## Problem

Modify the `search` method of the `BFSearch` class so that it does not support backtrack requests.

# Limit depth of search

JPF can be configured to limit the depth of the search by setting the JPF property `search.depth_limit`. The default value of `search.depth_limit` is `Integer.MAX_VALUE`. The `Search` class contains the attribute `depth` that can be used to keep track of the depth of the search. It also provides the method `getDepthLimit` which returns the maximal allowed depth of the search.

## Problem

Modify the `search` method of the `BFSearch` class to limit the depth.



# Limit memory usage

The JPF property `search.min_free` captures the minimal amount of memory, in bytes, that needs to remain free. The default value is  $1024 \ll 10$ . By leaving some memory free, JPF can report that it ran out of memory and provide some useful statistics instead of simply throwing an **OutOfMemoryError**. The method **checkStateSpaceLimit** of the class **Search** checks whether the minimal amount of memory that should be left free is still available.

## Problem

Modify the **search** method of the **BFSearch** class to limit the memory usage.

# Multiple errors?

The JPF property search.`multiple_errors` tells us whether the search should report multiple errors (or just the first one). The `forward` method also checks whether any property is violated after the unexplored transition has been traversed. If a violation has been detected then the attribute `done` is set to true if and only if JPF has been configured to report at most one error. The method `hasPropertyTermination` of the class `Search` checks whether a violation was encountered during the last transition. The method returns true if and only if a violation was encountered and the attribute `done` is set to true.

## Problem

Modify the `search` method of the `BFSearch` class to take `search.multiple_errors` into account.

A search should also notify listeners of particular events by invoking to the methods of the interface `SearchListener`, which can be found in the package `gov.nasa.jpf.search`. The `Search` class contains a number of `notify` methods.

## Problem

Modify the `search` method of the `BFSearch` class to incorporate notifications.