Search

EECS 4315

www.eecs.yorku.ca/course/4315/

Source: weknowyourdreams.com

Search strategies

JPF contains different search strategies:

- depth first search (`gov.nasa.jpf.search.DFSearch`),
- breadth first search (`gov.nasa.jpf.search.heuristic.BFSHeuristic`)
- and several other search strategies.

JPF has been designed in such a way that it can easily be extended. For example, a new search strategy can be added to JPF.

The Search class

The class `Search` of the package `gov.nasa.jpf.search` contains numerous attributes and methods that are useful for implementing search strategies.

By extending the `Search` class, we inherit all these features.

Depth first search

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

public class DFSearch extends Search {
  ...
}
```

Constructor of DFSearch

```
public Search(Config config, VM vm)
```

- The `Config` object contains the JPF properties.
- The `VM` object refers to JPF's virtual machine.

Question

Implement the constructor of the `DFSearch`.

The search method

The method

`public void search()`

drives the search.

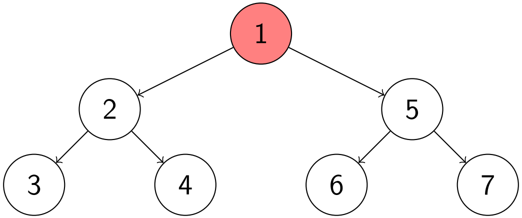
`public boolean forward()`

tries to move forward along an unexplored transition and returns whether the move is successful.

`public boolean backtrack()`

tries to backtrack and returns whether the backtrack is successful.

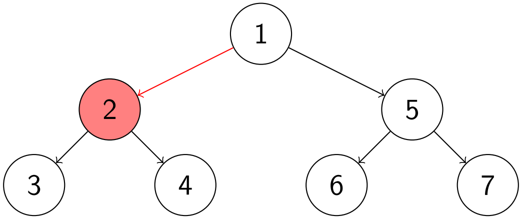
The search method



Question

For the above state space, provide the sequence of calls to `forward` and `backward` and the value returned by them corresponding to depth first search started in the top most state.

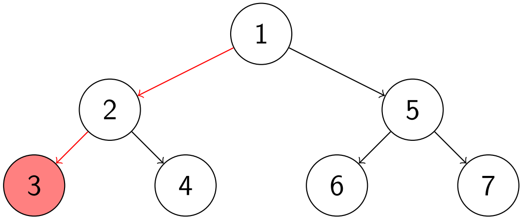
The search method



Answer

forward(true)

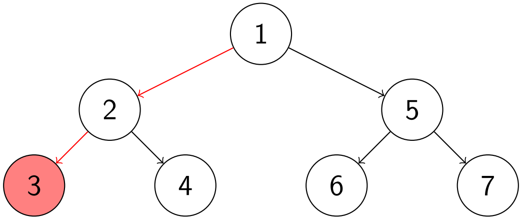
The search method



Answer

forward(true); forward(true)

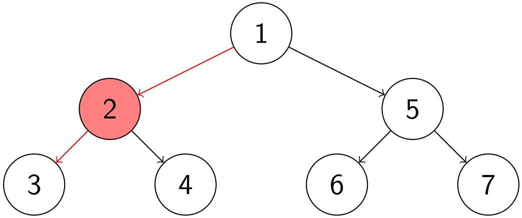
The search method



Answer

forward(true); forward(true); forward(false)

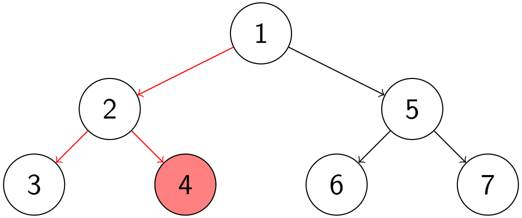
The search method



Answer

forward(true); forward(true); forward(false); backtrack(true)

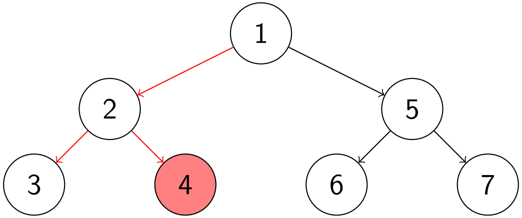
The search method



Answer

forward(true); forward(true); forward(false); backtrack(true); forward(true)

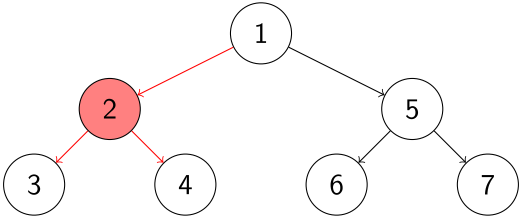
The search method



Answer

forward(true); forward(true); forward(false); backtrack(true); forward(true); forward(false)

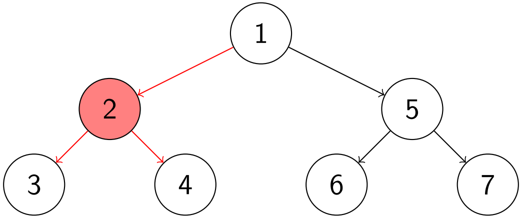
The search method



Answer

forward(true); forward(true); forward(false); backtrack(true);
forward(true); forward(false); backtrack(true)

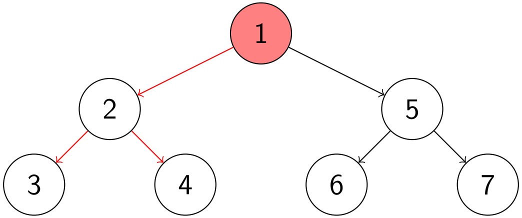
The search method



Answer

forward(true); forward(true); forward(false); backtrack(true);
forward(true); forward(false); backtrack(true); forward(false)

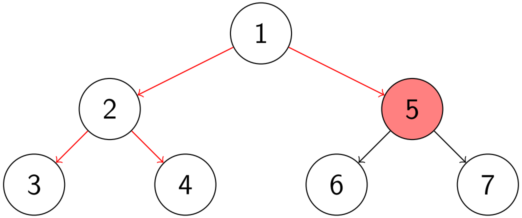
The search method



Answer

forward(true); forward(true); forward(false); backtrack(true); forward(true); forward(false); backtrack(true); forward(false); backtrack(true)

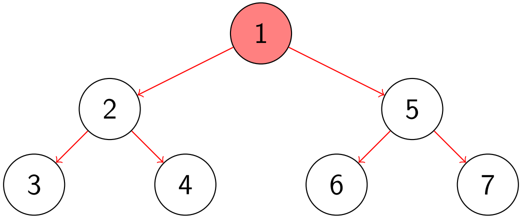
The search method



Answer

forward(true); forward(true); forward(false); backtrack(true); forward(true); forward(false); backtrack(true); forward(false); backtrack(true); forward(true)

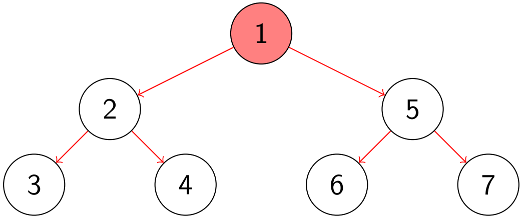
The search method



Answer

forward(true); forward(true); forward(false); backtrack(true); forward(true); forward(false); backtrack(true); forward(false); backtrack(true); forward(true); $\cdots$; forward(false)

The search method



Answer

forward(true); forward(true); forward(false); backtrack(true); forward(true); forward(false); backtrack(true); forward(false); backtrack(true); forward(true); $\cdots$; forward(false); backtrack(false)

The search method

Question

Write some code consisting only of calls to `forward` and `backward` and loops that gives rise to the sequence on the previous slide.

The search method

Question

Write some code consisting only of calls to `forward` and `backward` and loops that gives rise to the sequence on the previous slide.

Answer

There are many ways to express the sequence including

```
do {
  while (this.forward()) {}
} while (this.backtrack());
```

States

```
public boolean isNewState()
```

tests whether the current state has not been visited before.

States

`public boolean isNewState()`

tests whether the current state has not been visited before.

Question

Incorporate the `isNewState` method into the `search` method of the `DFSearch` class.

States

`public boolean isNewState()`

tests whether the current state has not been visited before.

Question

Incorporate the `isNewState` method into the `search` method of the `DFSearch` class.

Answer

```
do {
  while (this.forward() &&
         this.isNewState()) {}
} while (this.backtrack());
```

States

```
public boolean isEndState()
```

tests whether the current state is a final state.

States

`public boolean isEndState()`

tests whether the current state is a final state.

Question

Incorporate the `isEndState` method into the `search` method of the `DFSearch` class.

States

```
public boolean isEndState()
```

tests whether the current state is a final state.

Question

Incorporate the `isEndState` method into the `search` method of the `DFSearch` class.

Answer

```
do {
  while (this.forward() &&
         this.isNewState() &&
         !this.isEndState()) {}
} while (this.backtrack());
```

States

`public boolean isIgnoredState()`

tests whether the current state can be ignored in the search.

States can, for example, be ignored by using in the system under test the method `ignoreIf(boolean)` of JPF's class `Verify` which is part of the package `gov.nasa.jpf.vm`.

States

`public boolean isIgnoredState()`

tests whether the current state can be ignored in the search.

States can, for example, be ignored by using in the system under test the method `ignoreIf(boolean)` of JPF's class `Verify` which is part of the package `gov.nasa.jpf.vm`.

Question

Incorporate the `isIgnoredState` method into the `search` method of the `DFSearch` class.

States

`public boolean isIgnoredState()`

tests whether the current state can be ignored in the search.

States can, for example, be ignored by using in the system under test the method `ignoreIf(boolean)` of JPF's class `Verify` which is part of the package `gov.nasa.jpf.vm`.

Question

Incorporate the `isIgnoredState` method into the `search` method of the `DFSearch` class.

Answer

```
do {
  while (this.forward() &&
         this.isNewState() &&
         !this.isEndState() &&
         !this.isIgnoredState()) {}
} while (this.backtrack());
```

The done attribute

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

The done attribute

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

Question

Modify the `search` method of the `DFSearch` class to incorporate the `done` attribute.

The done attribute

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

Question

Modify the `search` method of the `DFSearch` class to incorporate the `done` attribute.

Answer

```
do {
  while (!this.done &&
         this.forward() &&
         this.isNewState() &&
         !this.isEndState() &&
         !this.isIgnoredState()) {}
} while (!this.done && this.backtrack());
```

Request backtrack

Other components of JPF can request a search to backtrack by means of the method

```
public boolean checkAndResetBacktrackRequest()
```

Request backtrack

Other components of JPF can request a search to backtrack by means of the method

```
public boolean checkAndResetBacktrackRequest()
```

Question

Modify the `search` method of the `DFSearch` class to incorporate the `checkAndResetBacktrackRequest` method.

Request backtrack

Other components of JPF can request a search to backtrack by means of the method

```
public boolean checkAndResetBacktrackRequest()
```

Question

Modify the `search` method of the `DFSearch` class to incorporate the `checkAndResetBacktrackRequest` method.

Answer

```
do {
  while (!this.done &&
         !this.checkAndResetBacktrackRequest() &&
         this.forward() &&
         this.isNewState() &&
         !this.isEndState() &&
         !this.isIgnoredState()) {}
} while (!this.done && this.backtrack());
```

Depth of search

The `Search` class contains the attribute `depth` that can be used to keep track of the depth of the search. It is initialized to zero.

Depth of search

The `Search` class contains the attribute `depth` that can be used to keep track of the depth of the search. It is initialized to zero.

Question

Override the `forward` method of the `Search` class to keep track of the depth.

Depth of search

The `Search` class contains the attribute `depth` that can be used to keep track of the depth of the search. It is initialized to zero.

Question

Override the `forward` method of the `Search` class to keep track of the depth.

Answer

```
protected boolean forward() {
  boolean successful = super.forward();
  if (successful) {
    this.depth++;
    }
  return successful;
}
```

Depth of search

The `Search` class contains the attribute `depth` that can be used to keep track of the depth of the search. It is initialized to zero.

Depth of search

The `Search` class contains the attribute `depth` that can be used to keep track of the depth of the search. It is initialized to zero.

Question

Override the `backtrack` method of the `Search` class to keep track of the depth.

Depth of search

The `Search` class contains the attribute `depth` that can be used to keep track of the depth of the search. It is initialized to zero.

Question

Override the `backtrack` method of the `Search` class to keep track of the depth.

Answer

```
protected boolean backtrack() {
  boolean successful = super.backtrack();
  if (successful) {
    this.depth--;
  }
  return successful;
}
```

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 provides the method getDepthLimit which returns the maximal allowed depth of the search.

We introduce the following method in the DFSearch class.

```
private boolean checkDepthLimit() {
  return this.depth < this.getDepthLimit();
}
```

Depth of search

Question

Incorporate `checkDepthLimit` into `forward`.

Depth of search

Question

Incorporate `checkDepthLimit` into `forward`.

Answer

```
protected boolean forward() {
  boolean successful = super.forward();
  if (successful) {
    this.depth++;
    successful = this.checkDepthLimit();
  }
  return successful;
}
```

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 $1024 \ll 10 = 1024^2 = 1,048,576B \approx 1MB$. 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.

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 $1024 \ll 10 = 1024^2 = 1,048,576B \approx 1MB$. 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.

Question

Modify the `search` method of the `DFSearch` class to limit the memory usage.

Limit memory usage

Answer

```
do {
  while (!this.done &&
         !this.checkAndResetBacktrackRequest() &&
         this.forward() &&
         !this.checkStateSpaceLimit() &&
         this.isNewState() &&
         !this.isEndState() &&
         !this.isIgnoredState()) {}
} while (!this.done &&
         !this.checkStateSpaceLimit() &&
         this.backtrack());
```

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.

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.

Question

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

Limit memory usage

Answer

```
do {
while (!this.done &&
       !this.checkAndResetBacktrackRequest() &&
       this.forward() &&
       !this.checkStateSpaceLimit() &&
       this.isNewState() &&
       !this.isEndState() &&
       !this.isIgnoredState() &&
       !this.hasPropertyTermination()) {}
} while (!this.done &&
         !this.checkStateSpaceLimit() &&
         this.backtrack());
```

Notifications

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.

Notifications

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.

Question

Modify the `search` method of the `DFSearch` class to incorporate following notifications.

- `notifySearchStarted`
- `notifySearchFinished`

Notifications

Answer

```
this.notifySearchStarted();
do {
  while (!this.done &&
         !this.checkAndResetBacktrackRequest() &&
         this.forward() &&
         !this.checkStateSpaceLimit() &&
         this.isNewState() &&
         !this.isEndState() &&
         !this.isIgnoredState() &&
         !this.hasPropertyTermination()) {}
} while (!this.done &&
         !this.checkStateSpaceLimit() &&
         this.backtrack());
this.notifySearchFinished();
```

Notifications

Question

Incorporate following notifications into the `forward` and `backtrack` method.

- `notifyStateAdvanced`
- `notifyStateBacktracked`
- `notifyStateProcessed`

Notifications

Answer

```
protected boolean forward() {
  boolean successful = super.forward();
  if (successful) {
    this.notifyStateAdvanced();
  } else {
    this.notifyStateProcessed();
  }
  return successful;
}

protected boolean backtrack() {
  boolean successful = super.backtrack();
  if (successful) {
    this.notifyStateBacktracked();
  }
  return successful;
}
```

Notifications

Question

Override the `checkStateSpaceLimit` method and modify the `checkDepthLimit` method to incorporate `notifySearchConstraintHit(String)` to notify the following.

- "memory limit reached"
- "depth limit reached"

Notifications

Answer

```
public boolean checkStateSpaceLimit() {
  boolean available = super.checkStateSpaceLimit();
  if (!available) {
    this.notifySearchConstraintHit("memory limit reached: "
  }
  return available;
}

private boolean checkDepthLimit() {
  boolean below = this.depth < this.getDepthLimit();
  if (!below) {
    this.notifySearchConstraintHit("depth limit reached: "
  }
  return below;
}
```

Notifications

Immediately after an invocation of the `forward` method of the `Search` class, an invocation of the `getCurrentError` method of the `Search` class returns `null` if and only if no property violation has been detected.

Notifications

Immediately after an invocation of the `forward` method of the `Search` class, an invocation of the `getCurrentError` method of the `Search` class returns `null` if and only if no property violation has been detected.

Question

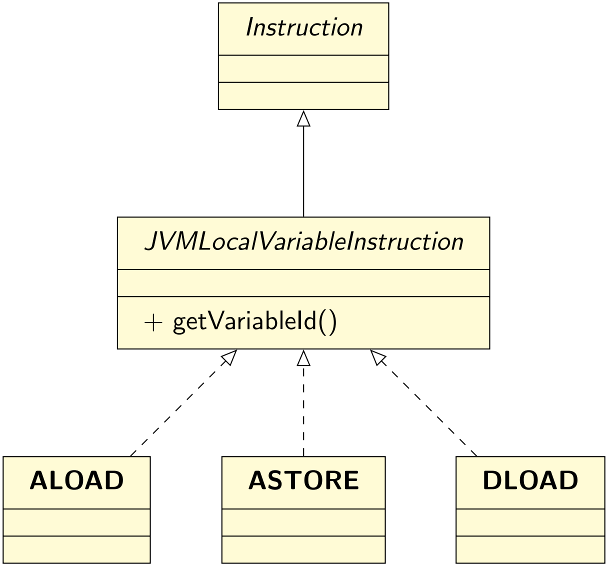
Modify the overridden `forward` method of the `DFSearch` class to include an invocation of the `notifyPropertyViolated` method.

Notifications

Answer

```
protected boolean forward() {
  boolean successful = super.forward();
  if (successful) {
    this.notifyStateAdvanced();
    if (this.getCurrentError() != null) {
      this.notifyPropertyViolated();
    }
  } else {
    this.notifyStateProcessed();
  }
  return successful;
}
```

Instruction classes



Print variable ID of an Instruction

Question

Does every `Instruction` object have a variable ID?

Print variable ID of an Instruction

Question

Does every `Instruction` object have a variable ID?

Answer

No, only `JVMLocalVariableInstruction` objects.

Print variable ID of an Instruction

Question

How do we limit our attention to `JVMLocalVariableInstruction` objects?

Print variable ID of an Instruction

Question

How do we limit our attention to `JVMLocalVariableInstruction` objects?

Answer

```
.... (Instruction instruction) {
  if (instruction instanceof JVMLocalVariableInstruction) {
    JVMLocalVariableInstruction variableInstruction =
      (JVMLocalVariableInstruction) instruction;
    System.out.println(variableInstruction.getVariableId());
  }
}
```

Print variable ID of an Instruction

Question

Can we use `getClass` instead of `instanceof`?

Print variable ID of an Instruction

Question

Can we use `getClass` instead of `instanceof`?

Answer

No. Note that `JVMLocalVariableInstruction` is an abstract class. Hence, one cannot create instances of the class. Therefore, `instruction.getClass() == JVMLocalVariableInstruction.class` always returns false. On the other hand, `instruction instanceof JVMLocalVariableInstruction` tests whether `instruction` is an instance of `JVMLocalVariableInstruction` or any of its subclasses.

Project

- January 25: install JPF (5%)
- February 15: draft proposal (2%)
- February 25: proposal (3%)
- March 8: first progress report (5%)
- March 22: second progress report (5%)
- Exam period: deliverables (20%)

Project

Very brief descriptions of the last three years' projects can be found here.

Students can work alone or in groups of two on their project.

Students are expected to work on average two hours per week on their projects from now on.

Project: potential topics

Potential topics include (but are not limited to)

- implement a new listener,
- implement a new search strategy,
- apply JPF to some nontrivial code with randomization or concurrency,
- improve an existing listener (by adding Javadoc, improving variable names, improving code structure, developing tests, etc),
- improve an existing search strategy (by adding Javadoc, improving variable names, improving code structure, developing tests, etc),
- add functionality to an existing listener, and
- add functionality to an existing search strategy.