1 Constructor

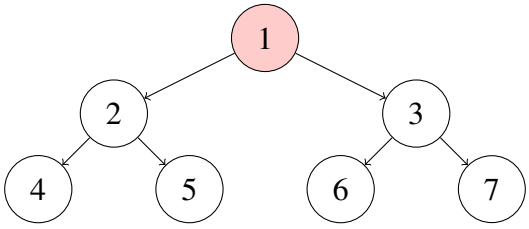
Implement the constructor of the **`BFSearch`** class.

```
public class BFSearch extends Search {
  /**
   * Initialize this search.
   *
   * @param config JPF's configuration.
   * @param vm JPF's virtual machine.
   */
  public BFSearch(Config config, VM vm) {



  }
```

2 Data structure

Which data structure is usually used to implement breadth first search?

3 Forward and backtrack



For the above state space, provide the content of the queue and the sequence of calls to **`forward`**, **`backtrack`**, **`enqueue`** and **`dequeue`**, and the value returned by the first two, corresponding to breadth first search started in the top most state. Assume that initially the queue contains the top most state.

4 Search

Implement a basic **`search`** method using only calls to **`forward`**, **`backtrack`**, **`enqueue`**, **`dequeue`** and **`isEmpty`** and loops.

```
public void search() {









}
```

5 Restorable states

We introduce the following methods.

```
/**
* Returns the current state so that it is restorable.
* @return the current state.
*/
private RestorableVMState getRestorableState() {
  return this.getVM().getRestorableState();
}

/**
* Restores the given state.
* @param state a state that is restorable.
*/
private void restoreState(RestorableVMState state) {
  this.getVM().restoreState(state);
}
```

Implement the **`search`** method using a **`Queue`** of **`RestorableVMState`**s.

```
public void search() {









}
```

6 New states

`public boolean isNewState()`

tests whether the current state has not been visited before.

Modify the **`search`** method so that each state is enqueued at most once.

```
public void search() {









}
```

7 End states

`public boolean isEndState()`

tests whether the current state is a final state.

Modify the **`search`** method so that end states are not enqueued.

```
public void search() {

}
```

8 Ignored 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`**.

Modify the **`search`** method so that ignored states are not enqueued.

```
public void search() {

}
```

9 Done

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

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

```
public void search() {

}
```

10 Request backtrack

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

`public boolean checkAndResetBacktrackRequest()`

Modify the **`BFSearch`** class so that it does not support backtrack requests.

11 Depth

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.

Modify the **`search`** method to keep track of the depth.

```
public void search() {









}
```

12 Depth limit

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 **BFSearch** class.

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

Incorporate **checkDepthLimit** into **search**.

```
public void search() {









}
```

13 Memory usage limit

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.

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

```
public void search() {









}
```