Software Engineering Final Exam

Design Document

Group 4

鄒咏霖・侯思岑・李辰暘・陳熙・李紹銘・汪宸宇

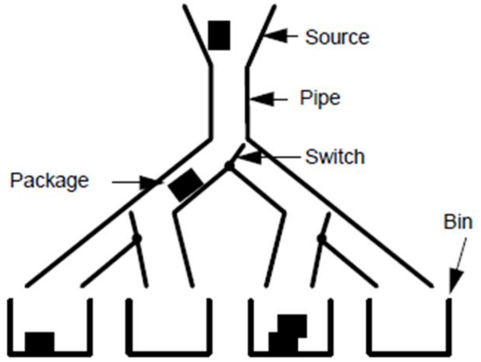
Outline

- Requirement
- Usecase Diagram
- Usecase Spec.
- Test Cases(Derive from Usecase)
- Class Diagram
- Implementation
  - Hint
  - Test Cases(For Implementation Testing)

Requirements Statement (I)

- A source station at the top feeds packages one at a time into the network, which is a binary tree consisting of switches connected by pipes. The terminal nodes of the binary tree are the destination bins.
- When a package arrives at the source station, its intended destination (one of the bins) is determined. The package is then released into the pipe leading from the source station. For a package to reach its designated destination bin, the switches in the network must be set to direct the package through the network and into the correct bin.

Requirements Statement (II)



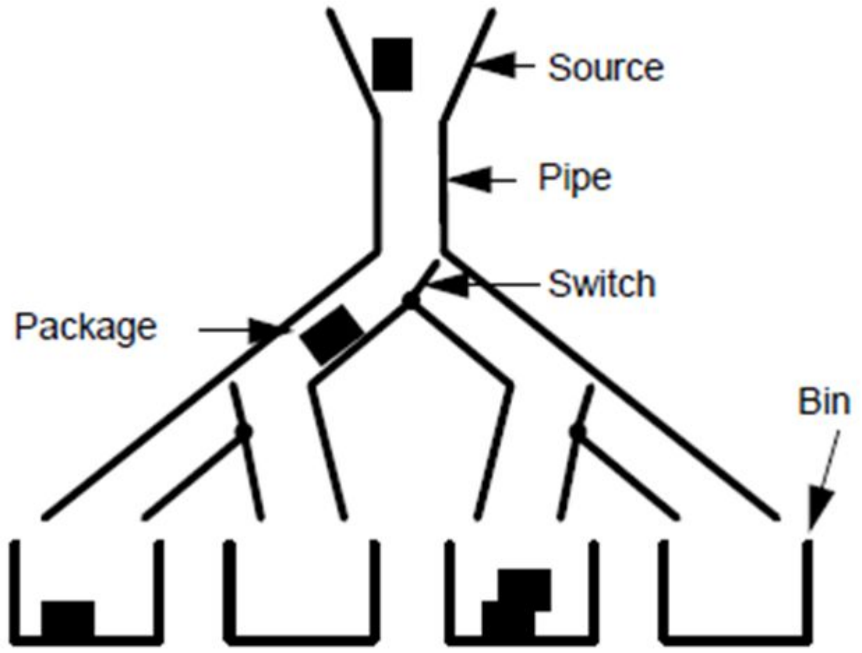
❑ Packages move through the network by gravity (working against friction), and so steady movement of packages cannot be guaranteed: they may bunch up within the network and thus make it impossible to set a switch properly between the passage of two such bunched packages (a switch cannot be set when there is a package or packages in the switch for fear of damaging such packages). If a new package's destination differs from that of the immediately preceding package, its release from the source station is delayed a pre-calculated, fixed length of time (to reduce the chance of bunching). In spite of such precautions, packages may still bunch up and become mis-routed, ending up in the wrong bin; the package router is to signal such an event.

Requirements Statement (III)

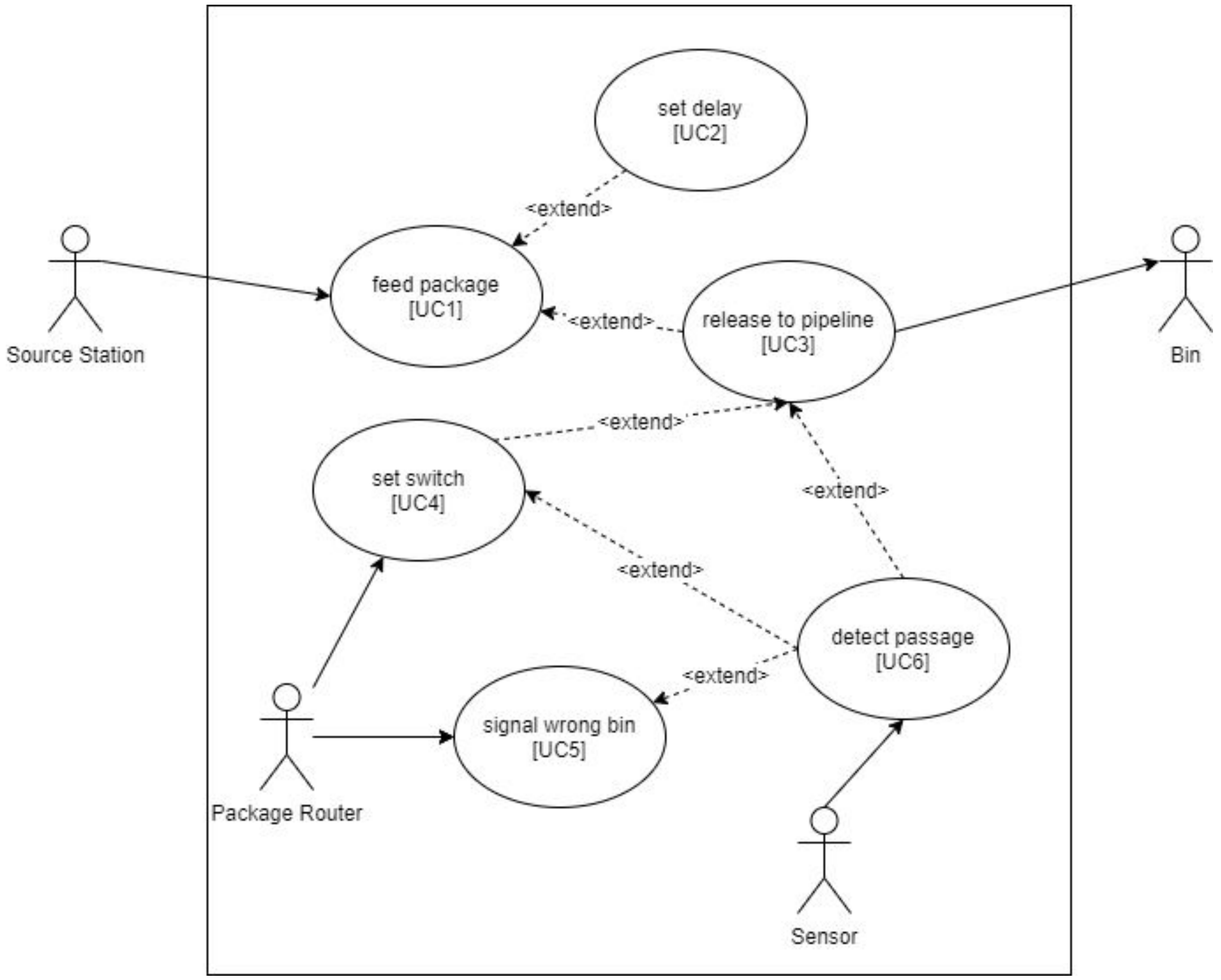
- Only a limited amount of information is available to the package router to effect its desired behavior. At the time of arrival at the source station but not thereafter, the destination of a package may be determined. The only means of determining the locations of packages within the network are sensors placed on the entries and exits of switches, and the entries of bins; these detect the passage of packages but are unable to determine their identity. (The sensors will be able to recognize the passage of individual packages, regardless of bunching).

Requirements Statement (IV)

The package router exemplar



Use Case Diagram



Actors:

- Source Station
- Bin
- Package Router
- Sensor

Use Case Specification

[UC1] Feed packages

- Requirements
  - (I) When a package arrives at the source station, its intended destination (one of the bins) is determined
  - (I) The package is then released into the pipe leading from the source station
  - (II) package, its release from the source station is delayed a pre-calculated, fixed length of time (to reduce the chance of bunching)
- Description
  - source station receives packages, and pass them to pipeline
- Actor: Source Station
- Basic Flow [BF1]
  1. packages arrive at the source station, with known destination bins
  2. source station set delay times, and apply delay strategy by extending [UC2]
  3. source station release packages into pipeline by including [UC3]
- Alternative Flow
  - 1.1 if package’s destination is invalid, ignore that package

[UC2] Set delay

- Requirements
  - (II) If a new package’s destination differs from that of the immediately preceding package, its release from the source station is delayed a pre-calculated, fixed length of time (to reduce the chance of bunching).
- Description
  - The package will be delayed a pre-calculated, fixed length of time if the package’s destination differs from the immediately preceding package.
- Actor
  - Source Station
- Basic Flow [BF2]
  - 1 If the arrived package’s destination is different from the preceding package, the package will be delayed for the pre-calculated, fixed length of time.
- Alternative Flow
  - 1.1 If the current package’s destination is the same as the preceding package, the package will not be delayed.

[UC3] Release to pipeline

- Requirements:
  - (I) When a package arrives at the source station, its intended destination (one of the bins) is determined. The package is then released into the pipe leading from the source station.
- Description: A package is released to the network, and the system tries to direct the package into the correct bin.
- Actor : Bin
- Basic Flow [BF3]
  1. A package with already determined destination is released into the pipe that leads from the source station.
  2. Extend [UC6] to enable sensor
  3. Extend [UC4] to enable switch
  4. The package arrives at the determined destination bin.
- Alternative Flow

[UC4] Set switch

- Requirements
  - (I) For a package to reach its designated destination bin, the switches in the network must be set to direct the package through the network and into the correct bin.
- Description
  - At certain time, system should display which switch is triggered and information of package bunber, [in/out] of the triggered switch and [L/R] when it is [out].
- Actor
  - Package router
- Basic Flow [BF4]
  1. There are some packages released to pipeline.
  2. Extending [UC6], each switch detects if there exists package(s) in a switch's upward pipe.
  3. If exists, then the switch set its operation.
- Alternative Flow
  - 2.1 if the detected package already mis-routed, then don't set any switch.
  - 2.2 if there are package(s) in the switch, then don't set any switch.

[UC5] Signal wrong bin

- Requirements
  - (II) Packages may still bunch up and become mis-routed, ending up in the wrong bin; The package router is to signal such an event.
- Description
  - Signal when the package ending up in the wrong bin
- Actor
  - package router
- Basic Flow [BF5]
  1. Monitoring sensors detecting the passage of packages on the entries of bins
  2. Find the sensor corresponding bin.
  3. Package router find the destionation bins of packages .
  4. If the destionation bins of packages are not same as the sensor corresponding bins, the package router signal wrong bin.
- Alternative Flow

[UC6] Detect passage

- Requirements
  - (III) sensors placed on the entries and exits of switches, and the entries of bins; these detect the passage of packages
- Description
  - While the system run, sensor in the system will continuously detecting any package that pass through it
- Actor
  - Sensor
- Basic Flow [BF6]

1. Detecting the pipe
2. If any package pass through, record it
3. back to step1

- Alternative Flow

Test Cases (Derived From Use Cases)

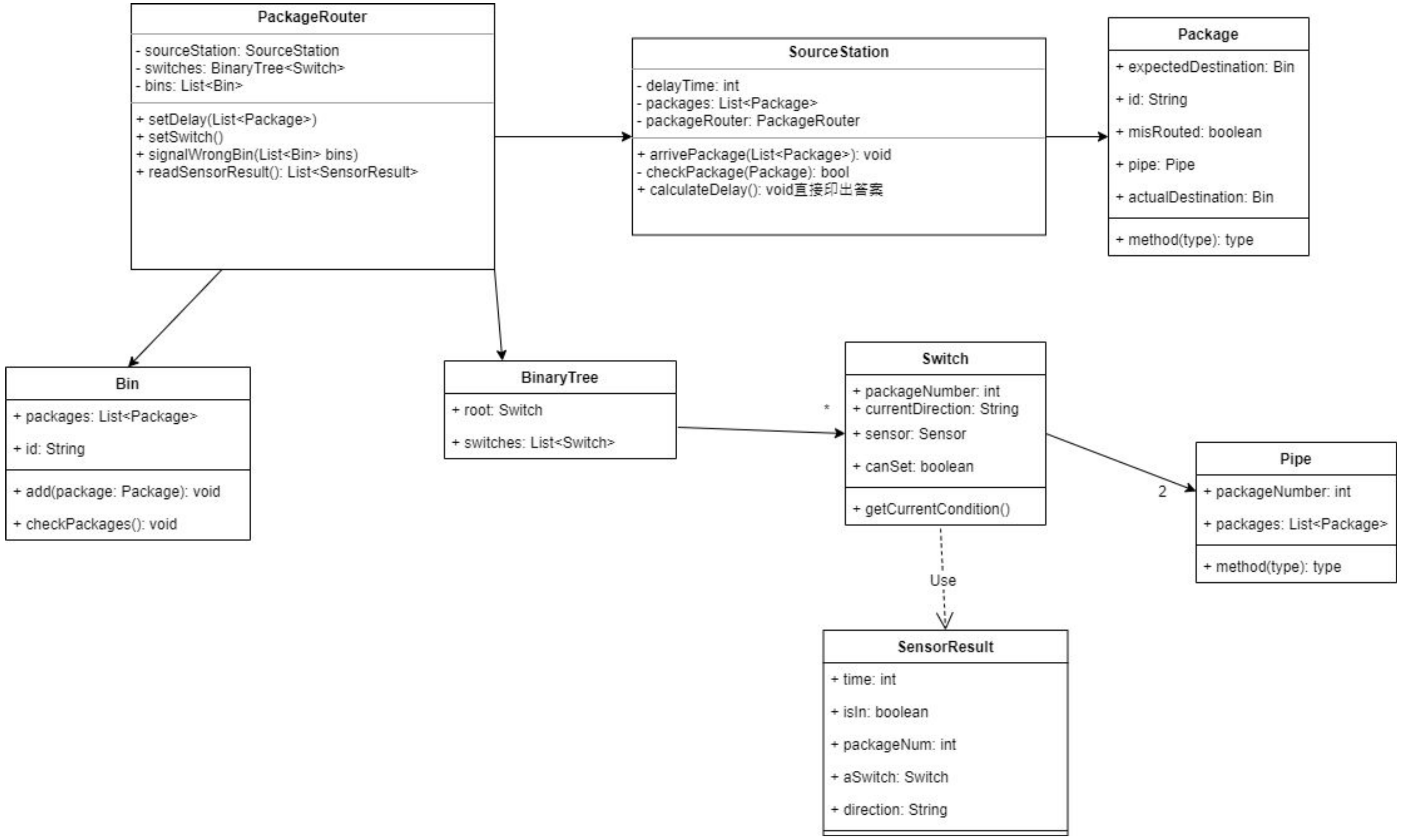
Scenarios

Scenario ID	Description	Flow	Ending State
S1	Current packages destination not valid	BF1 AF1.1.1	BF1
S2	Success routing with delay	BF1 BF2 BF3 BF6 BF4	END
S3	Success routing without delay	BF1 BF2 AF2.1.1 BF3 BF6 BF4	END
S4	Error routing because of bunch up	BF1 BF2 BF3 BF6 BF4 AF4.2.2	END
S5	Signal a misrouted event	BF1 BF2 BF3 BF6 BF4 AF4.2.1 BF5	END

Test Cases

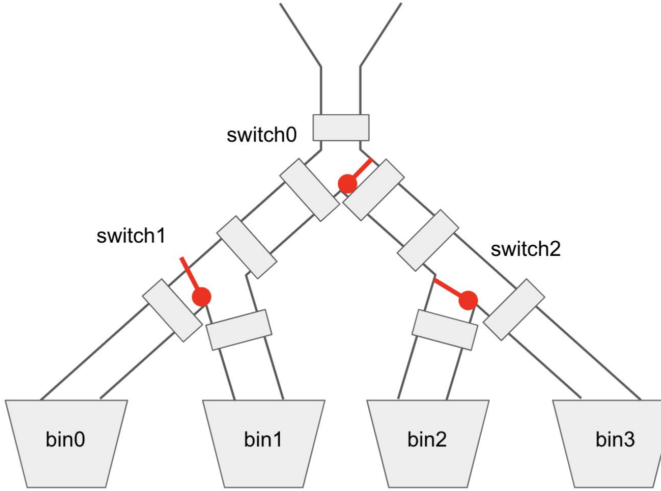
Scenario	Packages Destination	Destination Differ	Mis-routed	Bunch up	Expected Result
S1	Invalid	-	-	-	ignored it, back to BF1
S2	Valid	True	False	False	succesful route package to right bin
S3	Valid	False	False	False	succesful route package to right bin
S4	Valid	-	False	True	route package to wrong bin because of bunch up
S5	Valid	-	True	-	send an miss route message

Class Diagram


PackageRouter
- sourceStation: SourceStation
- switches: BinaryTree<Switch>
- bins: List<Bin>
+ setDelay(List<Package>)
+ setSwitch()
+ signalWrongBin(List<Bin> bins)
+ readSensorResult(): List<SensorResult>
SourceStation
- delayTime: int
- packages: List<Package>
- packageRouter: PackageRouter
+ arrivePackage(List<Package>): void
- checkPackage(Package): bool
+ calculateDelay(): void直接印出答案
Package
+ expectedDestination: Bin
+ id: String
+ misRouted: boolean
+ pipe: Pipe
+ actualDestination: Bin
+ method(type): type
Bin
+ packages: List<Package>
+ id: String
+ add(package: Package): void
+ checkPackages(): void
BinaryTree
+ root: Switch
+ switches: List<Switch>
*
Switch
+ packageNumber: int
+ currentDirection: String
+ sensor: Sensor
+ canSet: boolean
+ getCurrentCondition()
2
Pipe
+ packageNumber: int
+ packages: List<Package>
+ method(type): type
Use
SensorResult
+ time: int
+ isIn: boolean
+ packageNum: int
+ aSwitch: Switch
+ direction: String

Implementation

Hint



The Network is guaranteed to be a Perfect Binary Tree.

The output printing order should follows:

1. order by time

2. order by the sequence of any information read from input file

Hint For SCENARIO1 (Source station delays)

If the destination bin id of a package is invalid, just ignore the package and continue to process the next package.

Hint For SCENARIO2 (Set a switch)

1. You should set a switch even if it is in the same direction as the current one.
2. Do not set any switch for an already mis-routed package.
3. You should set a switch to guarantee that the first package that is not mis-routed in a bunch-up will move correctly.
4. You should set a switch as early as possible.
5. You can only set a switch if there is any package in the switch’s upward pipe.
6. You cannot set a switch if there is any package in it.

SCENARIO1: Source station delays

```
SCENARIO1 // tell program to execute scenario 1

[delay_time]

[bin_num]

[package_num]

[package_id] [destination bin_id of the package_id]

…
```

```
SCENARIO1
10
2
3
package0 bin0
package1 bin1
package2 bin1
```

```
[time]

[fed package_id1] [fed package_id2] ...

…
```

```
0
package0
10
package1 package2
```

For Scenario1:
If the destination bin id of a package is invalid, just ignore the package and continue to process the next package.

SCENARIO2: Set a switch

SCENARIO2 // tell program to execute scenario 2

[bin_num]

[package_num]

[package_id] [destination bin_id of the package_id]

…

[time] [in / out] [passage_num] [sourceStation / switch_id] ([L / R])

...

[time] set [switch_id] [L / R]

...

```
SCENARIO2
4
2
package0 bin0
package1 bin1
0 out 1 sourceStation
2 in 1 switch0
5 out 1 switch0 L
5 out 1 sourceStation
6 in 1 switch0
7 out 1 switch0 L
8 in 1 switch1
10 out 1 switch1 L
12 in 1 switch1
13 out 1 switch1 R
```

```
0 set switch0 L
5 set switch1 L
5 set switch0 L
10 set switch1 R
```

SCENARIO3: Signal Event

SCENARIO3 // tell program to execute scenario 3

[package_num]

[package_id] [destination bin_id of the package_id]

…

[bin_id] [received package_id1] [received package_id2] ...

...

[package_id] [wrong bin_id]

...

```
SCENARIO3
4
package0 bin0
package1 bin1
package2 bin2
package3 bin1
bin0 package2 package1
bin1 package3
bin2 package0
```

```
pacakge2 bin0
package1 bin0
package0 bin2
```

Test Cases (For Implementation Testing)

SCENARIO1 (i)

Test Case #1

INPUT

```
SCENARIO1
10
2
3
package0 bin0
package1 bin1
package2 bin1
```

OUTPUT

```
0
package0
10
package1 package2
```

Test Case #2

INPUT

```
SCENARIO1
11
4
6
p0 bin4
p1 bin1
p2 bin1
p3 bin3
p4 bin0
p5 bin0
```

OUTPUT

```
0
p0
11
p1 p2
22
p3
33
p4 p5
```

SCENARIO1 (ii)

Test Case #3

INPUT

```
SCENARIO1
7
4
5
package0 bin0
package1 bin2
package2 bin1
package3 bin3
package4 bin1
```

OUTPUT

```
0
package0
7
package1
14
package2
21
package3
28
package4
```

Test Case #4

INPUT

```
SCENARIO1
4
16
6
p0 bin11
p1 bin16
p2 bin9
p3 bin88
p4 bin9
p5 bin13
```

OUTPUT

```
0
p0
4
p2 p4
8
p5
```

SCENARIO2 (i)

Test Case #1

INPUT

```
SCENARIO2
4
2
package0 bin0
package1 bin1
0 out 1 sourceStation
2 in 1 switch0
5 out 1 switch0 L
5 out 1 sourceStation
6 in 1 switch0
7 out 1 switch0 L
8 in 1 switch1
10 out 1 switch1 L
12 in 1 switch1
13 out 1 switch1 R
```

OUTPUT

```
0 set switch0 L
5 set switch1 L
5 set switch0 L
10 set switch1 R
```

Test Case #2

INPUT

```
SCENARIO2
4
2
package0 bin1
package1 bin1
0 out 2 sourceStation
3 in 2 switch0
5 out 2 switch0 L
8 in 2 switch1
13 out 2 switch1 R
```

OUTPUT

```
0 set switch0 L
5 set switch1 R
```

SCENARIO2 (ii)

Test Case #3

INPUT

```
SCENARIO2
8
2
package0 bin0
package1 bin7
0 out 1 sourceStation
4 out 1 sourceStation
6 in 2 switch0
8 out 2 switch0 L
10 in 1 switch1
11 in 1 switch1
12 out 1 switch1 L
13 out 1 switch1 L
14 in 1 switch3
15 in 1 switch3
16 out 1 switch3 L
17 out 1 switch3 L
18 in 1 switch7
19 in 1 switch7
20 out 1 switch7 L
21 out 1 switch7 L
```

OUTPUT

```
0 set switch0 L
8 set switch1 L
12 set switch3 L
16 set switch7 L
```

SCENARIO3 (i)

Test Case #1

INPUT

```
SCENARIO3
4
package0 bin0
package1 bin1
package2 bin2
package3 bin1
bin0 package2 package1
bin1 package3
bin2 package0
```

OUTPUT

```
pacakge2 bin0
package1 bin0
package0 bin2
```

Test Case #2

INPUT

```
SCENARIO3
5
p0 bin1
p1 bin3
p2 bin3
p3 bin5
p4 bin15
bin1 p0
bin3 p2
bin5 p3 p1
bin15 p4
```

OUTPUT

```
p1 bin5
```

SCENARIO3 (ii)

Test Case #3

INPUT

```
SCENARIO3
3
package0 bin3
package1 bin10
package2 bin9
bin0 package1
bin7 package2
bin11 package0
```

OUTPUT

```
pacakge1 bin0
package2 bin7
package0 bin11
```

Test Case #4

INPUT

```
SCENARIO3
3
p0 bin1
p1 bin3
p2 bin3
bin1 p0
bin3 p1 p2
```

OUTPUT