DEPARTMENT OF INFORMATION TECHNOLOGY

HD in Software Engineering (IT114105) 2015/2016

ITP4510 DSA: Concepts and Implementation

<u>Cargo arrangement for delivery</u>

1. Problem Description

A train will collect cargo from *n* locations and rearrange them in the delivery center before delivery using another train. Assume each location will only have 1 cargo to deliver and the destination is labeled as 1 to n. The collector train collects the cargo in unsorted destination sequence. For example, the destination sequence collected may be as follows:

[5 8 1 7 4 2 9 6 3] 🚂 ← 9 cargos were collected in this example

destination no. - ‘3’ is the first cargo collected and with destination ‘3’

To facilitate efficient delivery of the cargos, cargos will be rearranged in the delivery center so that they are in the order 1 to n. When the cargos are in this order, the delivery train can visit the destination according to the pre-set sequence n to 1. The delivery sequence for the above example will be as follows:

[9 8 7 6 5 4 3 2 1] 🚂

delivery sequence – for destination‘1’ will be delivered LAST

Rearrangement of the cargos in the delivery center will be made via some holding buffer tracks (HBT). The number of HBT varies from 1 to n depending of their availability in the delivery center. Figure 1 below shows the cargos collection sequence (unsorted) with 3 HBTs (H1, H2 and H3).

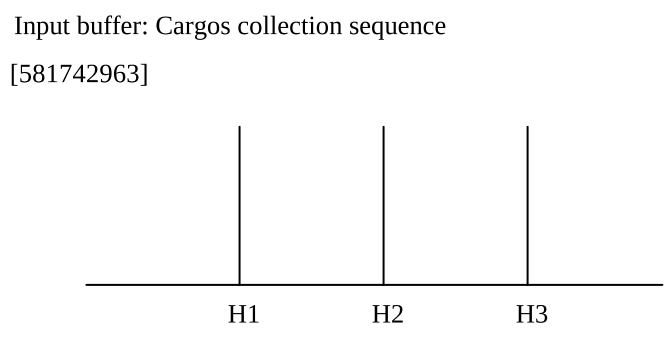


Figure 1

The *n* cargos are to end up in the delivery sequence (sorted) in the order 1 through n. According to the above example, figure 2 shows the 9 cargos in the desired sequence.

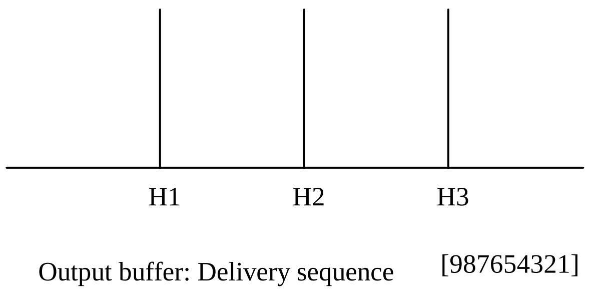


Figure 2

2. Solution Strategy: Using Stacks

We may also reconsider the cargos rearrangement problem with the holding buffer tracks (HBT) operate in a FILO (First In Last Out) manner and so may be regarded as stacks. When rearranging cargos, only the following moves are permitted:

2.1 The working principles

1. Move the cargos out from the input buffer one by one from the front (i.e. the train head position).

2. If a cargo moved out from the input buffer is not the next cargo to enter the output buffer because there are still higher numbered cargos not yet moved to the output buffer, it should be moved to the TOP of a suitable HBT.

3. The cargo must be moved to an empty HBT or the TOP of a HBT with the cargo in the original TOP has a higher number than the current moving cargo. If there is more than one suitable HBTs for the cargo, it should be moved to the HBT that has a lowest numbered TOP cargo among the HBTs (so that there is a higher chance for the next cargo to find a suitable holding track) and the empty HBT should be the last one to be considered.

4. If there is no suitable HBT, it means this is not possible to rearrange the cargos.

5. If a cargo moving out from the input buffer should be the next cargo to enter the output buffer because all the lower numbered cargos have been moved to the output buffer, this cargo should be moved to the output buffer directly.

6. Then, the TOP cargo of each HBT should be checked and see if it should be the next cargo to enter the output buffer or not (for the same reason, check and see if all the lowered numbered cargos have been moved to the output buffer or not). If yes, move it to the output buffer. This process of checking and moving of TOP cargo of a HBT to the output buffer is repeated until the TOP cargo of each HBT is not suitable to be moved to the output buffer.

2.2 Illustration of above example:

1. Cargo *3* is the front of the input buffer and cannot be output yet, as it is to be preceded by cargos *1* and *2*. So cargo *3* is detached and moved to the HBT H1.

2. The next cargo, cargo *6*, is also to be moved to a HBT. If cargo *6* is moved to H1, the rearrangement cannot be completed because cargo *3* will be below cargo *6*. However, cargo *3* is to be output before cargo *6* and so must leave H1 before cargo *6* does. So cargo *6* is put into H2.

3. The next cargo, cargo *9*, is put into H3 because putting it into either H1 or H2 will make it impossible to complete the rearrangement. Notice that whenever the cargo labels in a HBT are not in increasing order from top to bottom, the rearrangement cannot be completed.

4. Furthermore, cargo *2* can be pushed to HBTs H1, H2 or H3, but it is more appropriate to push it H1 because H1 has the smallest number top cargo among the 3 HBTs and it gives a better chance for the next cargo from the input buffer to enter H2 or H3.

5. Based on the previously mentioned rules and logic, the first 6 cargos will be moved to the 3 HBTs accordingly as shown in Figure 3.

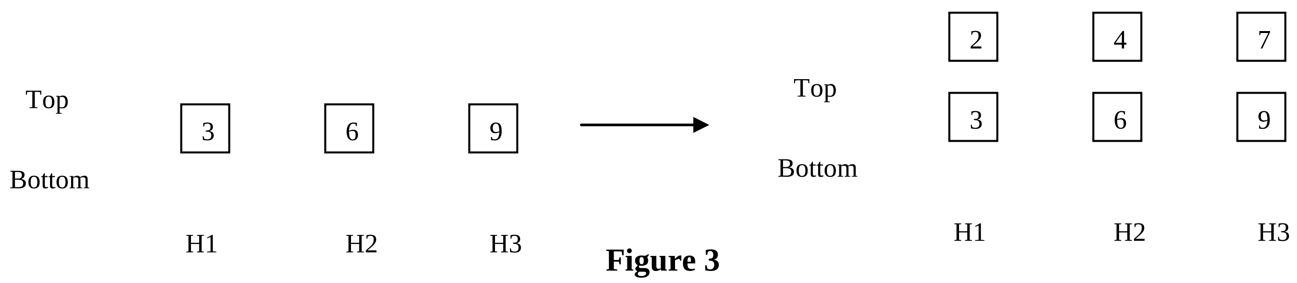
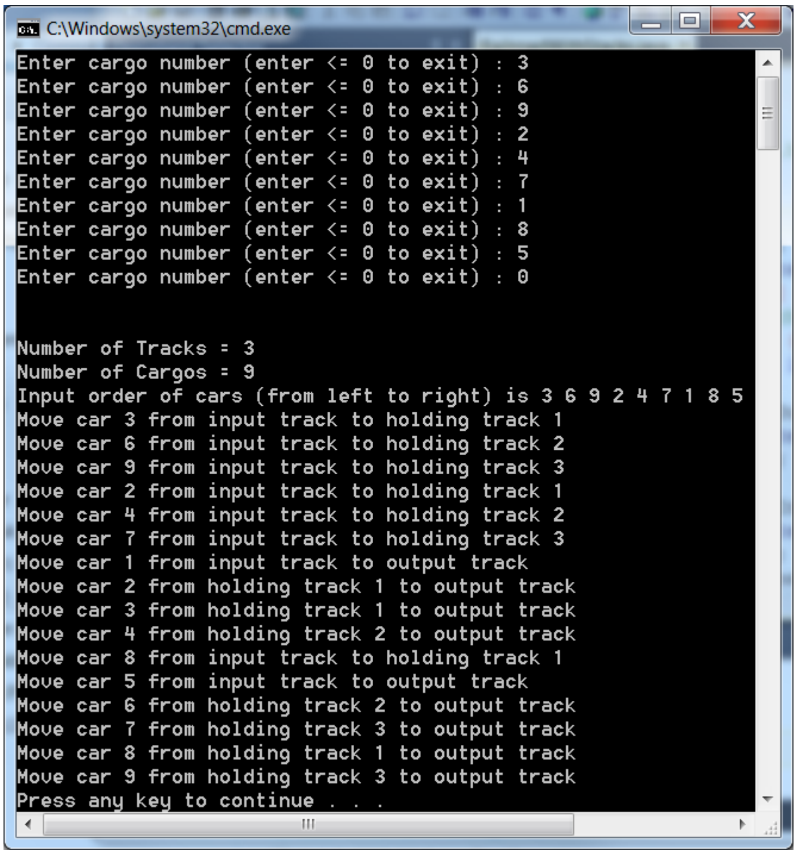


Figure 3

6. The next cargo, cargo *1* (i.e. the 7th cargo in the input buffer), can be moved to the output buffer directly.

7. Then, it is now the time to move cargo *2* from H1 to the output buffer. It is followed by moving cargo *3* from H1 to output buffer and cargo *4* from H2 to output buffer. No other cargo can be moved to the output buffer at this time. The next step is to move cargo *8* from input buffer to H1. Then cargo *5* is moved from the input buffer to the output buffer directly. Following this move, cargos *6*, *7*, *8* and *9* can be moved from the corresponding HBTs to the output buffer accordingly and the rearrangement is completed.

The following is a screen dump for sample run of the program with the above example:



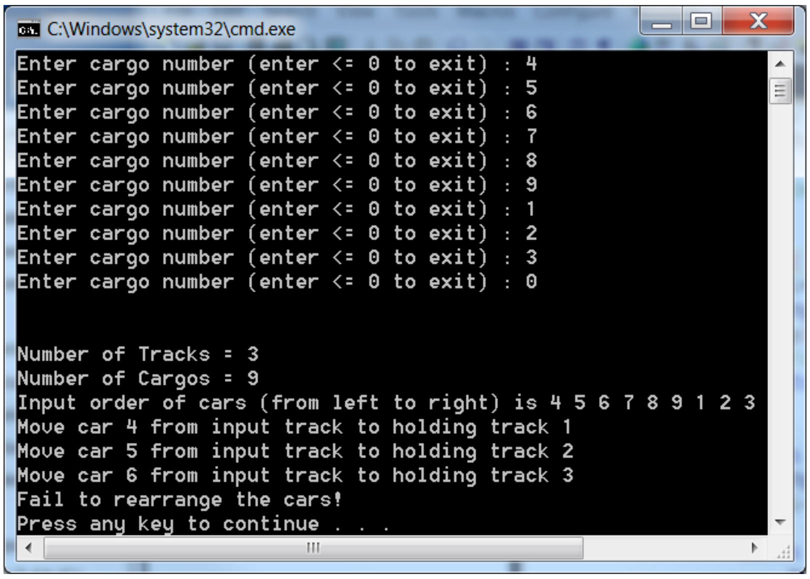
```
Enter cargo number (enter <= 0 to exit) : 3
Enter cargo number (enter <= 0 to exit) : 6
Enter cargo number (enter <= 0 to exit) : 9
Enter cargo number (enter <= 0 to exit) : 2
Enter cargo number (enter <= 0 to exit) : 4
Enter cargo number (enter <= 0 to exit) : 7
Enter cargo number (enter <= 0 to exit) : 1
Enter cargo number (enter <= 0 to exit) : 8
Enter cargo number (enter <= 0 to exit) : 5
Enter cargo number (enter <= 0 to exit) : 0


Number of Tracks = 3
Number of Cargos = 9
Input order of cars (from left to right) is 3 6 9 2 4 7 1 8 5
Move car 3 from input track to holding track 1
Move car 6 from input track to holding track 2
Move car 9 from input track to holding track 3
Move car 2 from input track to holding track 1
Move car 4 from input track to holding track 2
Move car 7 from input track to holding track 3
Move car 1 from input track to output track
Move car 2 from holding track 1 to output track
Move car 3 from holding track 1 to output track
Move car 4 from holding track 2 to output track
Move car 8 from input track to holding track 1
Move car 5 from input track to output track
Move car 6 from holding track 2 to output track
Move car 7 from holding track 3 to output track
Move car 8 from holding track 1 to output track
Move car 9 from holding track 3 to output track
Press any key to continue . . .
```

2.3 Unsuccessful case

If there are not enough HBTs for the rearrangement of cargos, the rearrangement process will arrive at a scenario that all TOP cargos of all HBTs have cargo number smaller than the current moving one in input buffer, and neither the current moving cargo nor any of the TOP cargos in the HBTs is the one to be moved to output buffer.   For example, if the input sequence is as follows:

[321987654]

After cargos 4, 5 and 6 were moved to tracks 1, 2 and 3, no more move can be made.



```
Enter cargo number (enter <= 0 to exit) : 4
Enter cargo number (enter <= 0 to exit) : 5
Enter cargo number (enter <= 0 to exit) : 6
Enter cargo number (enter <= 0 to exit) : 7
Enter cargo number (enter <= 0 to exit) : 8
Enter cargo number (enter <= 0 to exit) : 9
Enter cargo number (enter <= 0 to exit) : 1
Enter cargo number (enter <= 0 to exit) : 2
Enter cargo number (enter <= 0 to exit) : 3
Enter cargo number (enter <= 0 to exit) : 0


Number of Tracks = 3
Number of Cargos = 9
Input order of cars (from left to right) is 4 5 6 7 8 9 1 2 3
Move car 4 from input track to holding track 1
Move car 5 from input track to holding track 2
Move car 6 from input track to holding track 3
Fail to rearrange the cars!
Press any key to continue . . .
```

Instructions to Students

1. Assumption of the application:

   a. The number of holding buffer track is set to 3.

   b. The cargo number is input one by one until 0 or –ve value is entered.

   c. The cargo number MUST be from 1 to n without missing number and no checking is required.

   d. All cargo numbers are positive number and no duplication. Some examples are listed below:

      i. [2 4 3 1 5] ← valid

      ii. [6 A 3 1 2 4 5] ← should cause InputMismatchException

      Because number is entered one by one, it is suggested to use **try-catch** to handle this exception case and print error message before asking next input. [Hint: You can use the InputMismatchException class to handle this kind of exception].

      iii. [4 5 4 7 6 1 2 3] ← after getting the 2nd '4', DuplicatonException should be thrown

      Duplication error: a user-defined exception class should be created to handle this because no pre-defined exception class will handle this.

2. This assignment is an individual assignment and each student has to submit his/her own work. Plagiarism is a serious offence and any assignments that involve any plagiarism will be given ZERO marks. The award of Zero marks will apply to all parties, regardless of whether or not a student is the original author or the plagiarist. Further disciplinary action will follow.

3. You must use J2SDK 1.6.0 or above to develop the programs.

4. You are **NOT ALLOWED** to use data structure classes provided by Java API. **You must use the LinkedList, Stack (and/or other data structures developed in the labs)** but you may add new methods to the classes if necessary.

5. Your programs must follow the coding standard stated in Java coding standard published by Sun Microsystems (http://www.oracle.com/technetwork/java/codeconvtoc-136057.html). Marks will be deducted if the coding standard is not followed.

6. All work is to be submitted to Moodle on/before **1st April 2016 Friday 11:55pm**. Late submission without valid reasons will be given ZERO mark.

7. You are required to hand in

   (a) All classes (programs) written by you for this assignment.

   (b) The evidence of testing - You should submit <u>at least 4</u> dump screens to show the run results with your own input files (that means you should design your own test cases).

8. The weighting of this assignment is 40% of Continuous Assessment.

9. Marks allocation:

9.1	Solution strategy using Stacks	
	Implementation	64%
	Testing	12%
	Coding Standard	6%
	Appropriate Comments in programs	8%
9.2	Re-solve unsuccessful case	10%