



Banker's Algorithm

Rachatha Pramualsuk (5911694)



“The Banker's algorithm, sometimes referred to as the detection algorithm, is a resource allocation and deadlock avoidance algorithm developed by **Edsger Dijkstra** that tests for safety by simulating the allocation of predetermined maximum possible amounts of all resources, and then makes an "s-state" check to test for possible deadlock conditions for all other pending activities, before deciding whether allocation should be allowed to continue.”

—WIKIPEDIA

INTRODUCTION

The **key concept** is to find a 'safe **sequence**' in which after satisfying a process needs the system still maintain the ability to satisfy every other.

Purpose?


To prevent deadlock in a system multiple instances or resources.

How does it work?


Check one by one for a possible resource allocation then if safe, execute and release resource for other; else move on to other resource and check for a safe state.

Who came up with the algorithm?

Edsger Wybe Dijkstra - a Dutch computer scientist and an early pioneer in many research areas of computing science.



Using Kotlin
via Eclipse Java Neon



To code
Banker's Algorithm

```
BankersAlgo.kt Process.kt Resource.kt
1 package bankers
2
3 var resList: MutableList<Resource> = mutableListOf()
4 var procList: MutableList<Process> = mutableListOf()
5 var sum: MutableList<Int> = mutableListOf()
6 var nProcess = 0
7
8 fun main(args: Array<String>) {
9     init()
10    compute()
11 }
12
13 fun availableResource() {
14     print("Available Resource: ")
15     for (i in 0 until resList.size) {
16         print("${resList[i].available} ")
17     }
18     println()
19     println()
20 }
21
```

We will need to imitate the OS process and resource hence I created 2 lists:

```
BankersAlgo.kt Process.kt Resource.kt
1 package bankers
2
3 var resList: MutableList<Resource> = mutableListOf()
4 var procList: MutableList<Process> = mutableListOf()
5 var sum: MutableList<Int> = mutableListOf()
6 var nProcess = 0
7
8 fun main(args: Array<String>) {
9     init()
10    compute()
11 }
12
13 fun availableResource() {
14     print("Available Resource: ")
15     for (i in 0 until resList.size) {
16         print("${resList[i].available} ")
17     }
18     println()
19     println()
20 }
21
```

resList -> containing resource object
procList -> containing process object

```
BankersAlgo.kt Process.kt Resource.kt
1 package bankers
2
3 data class Resource(val name: String, var available: Int = 0, var max: Int = 0) {
4 }
```

```
BankersAlgo.kt Process.kt Resource.kt
1 package bankers
2
3 data class Process(val name: String) {
4     var taken: MutableList<Int> = mutableListOf()
5     var need: MutableList<Int> = mutableListOf()
6 }
```

```

1 package bankers
2
3 var resList: MutableList<Resource> = mutableListOf()
4 var procList: MutableList<Process> = mutableListOf()
5 var sum: MutableList<Int> = mutableListOf()
6 var nProcess = 0
7
8 fun main(args: Array<String>) {
9     init()
10    compute()
11 }
12
13 fun availableResource() {
14     print("Available Resource: ")
15     for (i in 0 until resList.size) {
16         print("${resList[i].available} ")
17     }
18     println()
19     println()
20 }
21

```

```

1 package bankers
2
3 data class Resource(val name: String, var available: Int = 0, var max: Int = 0)
4
5

```

```

1 package bankers
2
3 data class Process(val name: String) {
4     var taken: MutableList<Int> = mutableListOf()
5     var need: MutableList<Int> = mutableListOf()
6 }

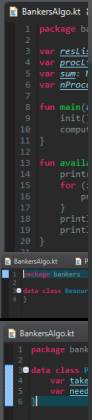
```

```

73 fun init() {
74
75     print("Number of resources: ")
76     val nResource = readLine()!!.toInt()
77
78     print("Number of processes: ")
79     nProcess = readLine()!!.toInt()
80     for (i in 0 until nProcess) {
81         procList.add(Process("P${i}"))
82     }
83     println()
84
85     print("Enter Maximum instances for each resource:\n")
86     for (i in 0 until nResource) {
87         resList.add(Resource(((65 + i).toChar()).toString()))
88     }
89     for (i in 0 until resList.size) {
90         print("${resList[i].name}'s max : ")
91         val inputMaximumResource = readLine()!!.toInt()
92         resList[i].max = inputMaximumResource
93     }
94     println()
95
96     print("Enter Allocated instances of resource for each process:\n")
97     for (i in 0 until procList.size) {
98         print("${procList[i].name}'s allocated : ")
99         val inputAllocatedRes = readLine()!!.split(" ")
100        for (j in 0 until inputAllocatedRes.size) {
101            procList[i].taken.add(inputAllocatedRes[j].toInt())
102        }
103    }
104    println()
105
106    print("Enter Maximum instances of resource required for each process:\n")
107    for (i in 0 until procList.size) {
108        print("${procList[i].name}'s needed : ")
109        val inputNeedRes = readLine()!!.split(" ")
110        for (j in 0 until inputNeedRes.size) {
111            procList[i].need.add(inputNeedRes[j].toInt())
112        }
113    }
114    println()
115
116    print("Allocated Resource: ")
117    for (i in 0 until resList.size) {
118        var count = 0
119        for (j in 0 until procList.size) {
120            count += procList[j].taken[i]
121        }
122        sum.add(count)
123        print("${sum[i]} ")
124    }
125    println()
126
127    print("Available Resource: ")
128    for (i in 0 until resList.size) {
129        resList[i].available = resList[i].max - sum[i]
130        print("${resList[i].available} ")
131    }

```

Initialize



```
73 fun init() {
74
75     print("Number of resources: ")
76     val nResource = readLine()!!.toInt()
77
78     print("Number of processes: ")
79     nProcess = readLine()!!.toInt()
80     for (i in 0 until nProcess) {
81         procList.add(Process("P${i}"))
82     }
83     println()
84
85     print("Enter Maximum instances for each resource:\n")
86     for (i in 0 until nResource) {
87         resList.add(Resource(((65 + i).toChar()).toString()))
88     }
89     for (i in 0 until resList.size) {
90         print("${resList[i].name}'s max : ")
91         val inputMaximumResource = readLine()!!.toInt()
92         resList[i].max = inputMaximumResource
93     }
94     println()
95
96     print("Enter Allocated instances of resource for each process:\n")
97     for (i in 0 until procList.size) {
98         print("${procList[i].name}'s allocated : ")
99         val inputAllocatedRes = readLine()!!.split(" ")
100         for (j in 0 until inputAllocatedRes.size) {
101             procList[i].taken.add(inputAllocatedRes[j].toInt())
102         }
103     }
104     println()
105
106     print("Enter Maximum instances of resource required for each process:\n")
107     for (i in 0 until procList.size) {
108         print("${procList[i].name}'s needed : ")
109         val inputNeedRes = readLine()!!.split(" ")
110         for (j in 0 until inputNeedRes.size) {
111             procList[i].need.add(inputNeedRes[j].toInt())
112         }
113     }
```

We prompt for inputs

```

1 package bank
2
3 var resList = ArrayList<Resource>()
4 var procl = ArrayList<Process>()
5 var sum = 0
6 var rProc = 0
7
8 fun main() {
9     init()
10    compute()
11 }
12
13 fun avail() {
14     print("Available Resource: ")
15     for (i in 0 until resList.size) {
16         print("${resList[i].name} ")
17     }
18     print()
19 }
20
21 fun take() {
22     print("Enter Allocated instances of resource for each process:\n")
23     for (i in 0 until procl.size) {
24         print("${procl[i].name}'s allocated : ")
25         val inputAllocatedRes = readLine()!!.split(" ")
26         for (j in 0 until inputAllocatedRes.size) {
27             procl[i].taken.add(inputAllocatedRes[j].toInt())
28         }
29     }
30     println()
31
32     print("Enter Maximum instances of resource required for each process:\n")
33     for (i in 0 until procl.size) {
34         print("${procl[i].name}'s needed : ")
35         val inputNeedRes = readLine()!!.split(" ")
36         for (j in 0 until inputNeedRes.size) {
37             procl[i].need.add(inputNeedRes[j].toInt())
38         }
39     }
40     println()
41
42     print("Allocated Resource: ")
43     for (i in 0 until resList.size) {
44         var count = 0
45         for (j in 0 until procl.size) {
46             count += procl[j].taken[i]
47         }
48         sum.add(count)
49         print("${sum[i]} ")
50     }
51     println()
52
53     print("Available Resource: ")
54     for (i in 0 until resList.size) {
55         resList[i].available = resList[i].max - sum[i]
56         print("${resList[i].available} ")
57     }
58     println()
59     println()
60 }
61
62 }

```

We prompt for inputs

```
BankersAlgo.kt Process.kt Resource.kt
1 package bankers
2
3 var resList: MutableList<Resource> = mutableListOf()
4 var procList: MutableList<Process> = mutableListOf()
5 var sum: MutableList<Int> = mutableListOf()
6 var nProcess = 0
7
8 fun main(args: Array<String>) {
9     init()
10    compute()
11}
12
13 fun availableResource() {
14    print("Available Resource: ")
15    for (i in 0 until resList.size) {
16        print("${resList[i].available} ")
17    }
18    println()
19    println()
20}
21
22 package bankers
23
24 data class Resource(val name: String, var available: Int = 0, var max: Int = 0)
25
26 package bankers
27
28 fun init() {
29    print("Number of resources: ")
30    val nResource = readLine()!!.toInt()
31
32    print("Number of processes: ")
33    nProcess = readLine()!!.toInt()
34    for (i in 0 until nProcess) {
35        procList.add(Process("P${i}"))
36    }
37    println()
38}
```

Number of resources: 4

Number of processes: 3

Enter Maximum instances for each resource:

A's max : 6

B's max : 5

C's max : 7

D's max : 6

Enter Allocated instances of resource for each process:

P0's allocated : 1 2 2 1

P1's allocated : 1 0 3 3

P2's allocated : 1 2 1 0

Enter Maximum instances of resource required for each process:

P0's needed : 2 1 0 1

P1's needed : 0 2 0 1

P2's needed : 0 1 4 0

|

```
121    }
122    sum.add(count)
123    print("${sum[i]} ")
124    }
125    println()
126
127    print("Available Resource: ")
128    for (i in 0 until resList.size) {
129        resList[i].available = resList[i].max - sum[i]
130        print("${resList[i].available} ")
131    }
```

In this format

```
1 package bankers
2
3 var reslist: MutableList<Resource> = mutableListOf()
4 var pprocess: MutableList<Process> = mutableListOf()
5 var sum: MutableList<Int> = mutableListOf()
6 var rProcess = 0
7
8 fun main(args: Array<String>) {
9     init()
10    compute()
11 }
12
13 fun availableResource() {
14    print("Available Resource: ")
15    for (i in 0 until reslist.size) {
16        print("${reslist[i].available} ")
17    }
18    println()
19    println()
20 }
21
22 // BankersAlgorithm
23 // package bankers
24 // data class Resource(val name: String, var available: Int = 0, var max: Int = 0) {
25 // }
26
27 // BankersAlgorithm
28 // package bankers
29 // data class Process(val name: String) {
30 //     var taken: MutableList<Int> = mutableListOf()
31 //     var need: MutableList<Int> = mutableListOf()
32 // }
33
34 // BankersAlgorithm
35 // fun init() {
36 //     print("Number of resources: ")
37 //     val rResource = readLine()!!.toInt()
38 //     print("Number of processes: ")
39 //     rProcess = readLine()!!.toInt()
40 //     for (i in 0 until rProcess) {
41 //         pprocess.add(Process("P${i+1}"))
42 //     }
43 //     println()
44 //     print("Enter Maximum instances for each resource:\n")
45 //     for (i in 0 until rResource) {
46 //         reslist.add(Resource((i+1).toString(), toInt(), toInt()))
47 //     }
48 //     for (i in 0 until reslist.size) {
49 //         print("${reslist[i].name}'s max : ")
50 //         val tempMaxResource = readLine()!!.toInt()
51 //         reslist[i].max = inputMaxResource
52 //     }
53 //     println()
54 //     print("Enter Allocated instances of resource for each process:\n")
55 //     for (i in 0 until pprocess.size) {
56 //         print("${pprocess[i].name}'s allocated : ")
57 //         val tempAllocatedMax = readLine()!!.split(" ")
58 //         for (j in 0 until tempAllocatedMax.size) {
59 //             pprocess[i].taken.add(tempAllocatedMax[j].toInt())
60 //         }
61 //     }
62 //     println()
63 //     print("Enter Maximum instances of resource required for each process:\n")
64 //     for (i in 0 until pprocess.size) {
65 //         print("${pprocess[i].name}'s needed : ")
66 //         val tempNeededMax = readLine()!!.split(" ")
67 //         for (j in 0 until tempNeededMax.size) {
68 //             pprocess[i].need.add(tempNeededMax[j].toInt())
69 //         }
70 //     }
71 //     println()
72 //     print("Allocated Resource: ")
73 //     for (i in 0 until reslist.size) {
74 //         var count = 0
75 //         for (j in 0 until pprocess.size) {
76 //             count += pprocess[j].taken[i]
77 //         }
78 //         sum.add(count)
79 //         print("${sum[i]} ")
80 //     }
81 //     println()
82 //     print("Available Resource: ")
83 //     for (i in 0 until reslist.size) {
84 //         reslist[i].available = reslist[i].max - sum[i]
85 //         print("${reslist[i].available} ")
86 //     }
87 //     println()
88 //     println()
89 // }
90
91 Number of resources: 4
92 Number of processes: 3
93
94 Enter Maximum instances for each resource:
95 A's max : 6
96 B's max : 5
97 C's max : 7
98 D's max : 6
99
100 Enter Allocated instances of resource for each process:
101 P0's allocated : 1 2 2 1
102 P1's allocated : 1 0 3 3
103 P2's allocated : 1 2 1 0
104
105 Enter Maximum instances of resource required for each process:
106 P0's needed : 2 1 0 1
107 P1's needed : 0 2 0 1
108 P2's needed : 0 1 4 0
109 }
```

Then we execute the program

```
1 package bankers
2
3 var reslist: MutableList<Resource> = mutableListOf()
4 var avail: MutableList<Process> = mutableListOf()
5 var sum: MutableList<Int> = mutableListOf()
6 var nProcess = 0
7
8 fun main(args: Array<String>) {
9     init()
10    compute()
11 }
12
13 fun availableResource() {
14    print("Available Resource: ")
15    for (i in 0 until reslist.size) {
16        print("${reslist[i].available} ")
17    }
18    println()
19    println()
20 }
21
22 // BankersAlgorithm
23 // package bankers
24 // data class Resource(val name: String, var available: Int = 0, var max: Int = 0) {
25 // }
26
27 // BankersAlgorithm
28 // package bankers
29 // data class Process(val name: String) {
30 //     var taken: MutableList<Int> = mutableListOf()
31 //     var need: MutableList<Int> = mutableListOf()
32 // }
33
34 // BankersAlgorithm
35 // package bankers
36 // fun init() {
37 //     print("Number of resources: ")
38 //     val nResource = readLine()!!.toInt()
39 //     print("Number of processes: ")
40 //     nProcess = readLine()!!.toInt()
41 //     for (i in 0 until nProcess) {
42 //         avail.add(Process("P${i}"))
43 //     }
44 //     print()
45 //     print("Enter Maximum instances for each resource:\n")
46 //     for (i in 0 until nResource) {
47 //         reslist.add(Resource((i+1).toString(), 0, 0))
48 //     }
49 //     for (i in 0 until reslist.size) {
50 //         print("${reslist[i].name}'s max : ")
51 //         val tempMaxResource = readLine()!!.toInt()
52 //         reslist[i].max = tempMaxResource
53 //     }
54 //     print()
55 //     print("Enter Allocated instances of resource for each process:\n")
56 //     for (i in 0 until avail.size) {
57 //         print("${avail[i].name}'s allocated : ")
58 //         val tempAllocated = readLine()!!.split(" ")
59 //         for (j in 0 until tempAllocated.size) {
60 //             avail[i].taken.add(tempAllocated[j].toInt())
61 //         }
62 //     }
63 //     print()
64 //     print("Enter Maximum instances of resource required for each process:\n")
65 //     for (i in 0 until avail.size) {
66 //         print("${avail[i].name}'s needed : ")
67 //         val tempNeeded = readLine()!!.split(" ")
68 //         for (j in 0 until tempNeeded.size) {
69 //             avail[i].need.add(tempNeeded[j].toInt())
70 //         }
71 //     }
72 //     print()
73 //     print("Allocated Resource: ")
74 //     for (i in 0 until reslist.size) {
75 //         var count = 0
76 //         for (j in 0 until avail.size) {
77 //             count += avail[j].taken[i]
78 //         }
79 //         sum.add(count)
80 //         print("${sum[i]} ")
81 //     }
82 //     print()
83 //     print("Available Resource: ")
84 //     for (i in 0 until reslist.size) {
85 //         reslist[i].available = reslist[i].max - sum[i]
86 //         print("${reslist[i].available} ")
87 //     }
88 //     print()
89 //     print()
90 // }
91
92 // Number of resources: 4
93 // Number of processes: 3
94
95 // Enter Maximum instances for each resource:
96 // A's max : 6
97 // B's max : 5
98 // C's max : 7
99 // D's max : 6
100
101 // Enter Allocated instances of resource for each process:
102 // P0's allocated : 1 2 2 1
103 // P1's allocated : 1 0 3 3
104 // P2's allocated : 1 2 1 0
105
106 // Enter Maximum instances of resource required for each process:
107 // P0's needed : 2 1 0 1
108 // P1's needed : 0 2 0 1
109 // P2's needed : 0 1 4 0
```

Allocated Resource: 3 4 6 4

Available Resource: 3 1 1 2

Executing process P0

Available Resource: 4 3 3 3

Executing process P2

Available Resource: 5 5 4 3

Executing process P1

Available Resource: 6 5 7 6

Safe Sequence:

< P0 P2 P1 >

Getting this result

```

1 package bankers
2
3 var resList: MutableList<Resource> = mutableListOf()
4 var procList: MutableList<Process> = mutableListOf()
5 var sum: MutableList<Int> = mutableListOf()
6 var nProcess = 0
7
8 fun main(args: Array<String>) {
9     init()
10    compute()
11 }
12
13 fun availableResource() {
14    print("Available Resource: ")
15    for (i in 0 until resList.size) {
16        print("${resList[i].available} ")
17    }
18    println()
19    println()
20 }

```

BankersAlgo.kt Process.kt Resource.kt

BankersAlgo.kt Process.kt Resource.kt

```

1 package bankers
2
3 data class Resource(val name: String, var available: Int = 0, var max: Int = 0) {
4     // ...
5 }

```

BankersAlgo.kt Process.kt Resource.kt

```

1 package bankers
2
3 data class Process(val name: String) {
4     var taken: MutableList<Int> = mutableListOf()
5     var need: MutableList<Int> = mutableListOf()
6 }

```

BankersAlgo.kt Process.kt Resource.kt

```

1 fun init() {
2     // ...
3 }

```

BankersAlgo.kt Process.kt Resource.kt

```

1 // ...
2 // ...
3 // ...
4 // ...
5 // ...
6 // ...
7 // ...
8 // ...
9 // ...
10 // ...
11 // ...
12 // ...
13 // ...
14 // ...
15 // ...
16 // ...
17 // ...
18 // ...
19 // ...
20 // ...
21 // ...
22 // ...
23 // ...
24 // ...
25 // ...
26 // ...
27 // ...
28 // ...
29 // ...
30 // ...
31 // ...
32 // ...
33 // ...
34 // ...
35 // ...
36 // ...
37 // ...
38 // ...
39 // ...
40 // ...
41 // ...
42 // ...
43 // ...
44 // ...
45 // ...
46 // ...
47 // ...
48 // ...
49 // ...
50 // ...
51 // ...
52 // ...
53 // ...
54 // ...
55 // ...
56 // ...
57 // ...
58 // ...
59 // ...
60 // ...
61 // ...
62 // ...
63 // ...
64 // ...
65 // ...
66 // ...
67 // ...
68 // ...
69 // ...
70 // ...
71 // ...
72 // ...
73 // ...
74 // ...
75 // ...
76 // ...
77 // ...
78 // ...
79 // ...
80 // ...
81 // ...
82 // ...
83 // ...
84 // ...
85 // ...
86 // ...
87 // ...
88 // ...
89 // ...
90 // ...
91 // ...
92 // ...
93 // ...
94 // ...
95 // ...
96 // ...
97 // ...
98 // ...
99 // ...
100 // ...
101 // ...
102 // ...
103 // ...
104 // ...
105 // ...
106 // ...
107 // ...
108 // ...
109 // ...
110 // ...
111 // ...
112 // ...
113 // ...
114 // ...
115 // ...
116 // ...
117 // ...
118 // ...
119 // ...
120 // ...
121 // ...
122 // ...
123 // ...
124 // ...
125 // ...
126 // ...
127 // ...
128 // ...
129 // ...
130 // ...
131 // ...
132 // ...
133 // ...
134 // ...
135 // ...
136 // ...
137 // ...
138 // ...
139 // ...
140 // ...
141 // ...
142 // ...
143 // ...
144 // ...
145 // ...
146 // ...
147 // ...
148 // ...
149 // ...
150 // ...
151 // ...
152 // ...
153 // ...
154 // ...
155 // ...
156 // ...
157 // ...
158 // ...
159 // ...
160 // ...
161 // ...
162 // ...
163 // ...
164 // ...
165 // ...
166 // ...
167 // ...
168 // ...
169 // ...
170 // ...
171 // ...
172 // ...
173 // ...
174 // ...
175 // ...
176 // ...
177 // ...
178 // ...
179 // ...
180 // ...
181 // ...
182 // ...
183 // ...
184 // ...
185 // ...
186 // ...
187 // ...
188 // ...
189 // ...
190 // ...
191 // ...
192 // ...
193 // ...
194 // ...
195 // ...
196 // ...
197 // ...
198 // ...
199 // ...
200 // ...
201 // ...
202 // ...
203 // ...
204 // ...
205 // ...
206 // ...
207 // ...
208 // ...
209 // ...
210 // ...
211 // ...
212 // ...
213 // ...
214 // ...
215 // ...
216 // ...
217 // ...
218 // ...
219 // ...
220 // ...
221 // ...
222 // ...
223 // ...
224 // ...
225 // ...
226 // ...
227 // ...
228 // ...
229 // ...
230 // ...
231 // ...
232 // ...
233 // ...
234 // ...
235 // ...
236 // ...
237 // ...
238 // ...
239 // ...
240 // ...
241 // ...
242 // ...
243 // ...
244 // ...
245 // ...
246 // ...
247 // ...
248 // ...
249 // ...
250 // ...
251 // ...
252 // ...
253 // ...
254 // ...
255 // ...
256 // ...
257 // ...
258 // ...
259 // ...
260 // ...
261 // ...
262 // ...
263 // ...
264 // ...
265 // ...
266 // ...
267 // ...
268 // ...
269 // ...
270 // ...
271 // ...
272 // ...
273 // ...
274 // ...
275 // ...
276 // ...
277 // ...
278 // ...
279 // ...
280 // ...
281 // ...
282 // ...
283 // ...
284 // ...
285 // ...
286 // ...
287 // ...
288 // ...
289 // ...
290 // ...
291 // ...
292 // ...
293 // ...
294 // ...
295 // ...
296 // ...
297 // ...
298 // ...
299 // ...
300 // ...
301 // ...
302 // ...
303 // ...
304 // ...
305 // ...
306 // ...
307 // ...
308 // ...
309 // ...
310 // ...
311 // ...
312 // ...
313 // ...
314 // ...
315 // ...
316 // ...
317 // ...
318 // ...
319 // ...
320 // ...
321 // ...
322 // ...
323 // ...
324 // ...
325 // ...
326 // ...
327 // ...
328 // ...
329 // ...
330 // ...
331 // ...
332 // ...
333 // ...
334 // ...
335 // ...
336 // ...
337 // ...
338 // ...
339 // ...
340 // ...
341 // ...
342 // ...
343 // ...
344 // ...
345 // ...
346 // ...
347 // ...
348 // ...
349 // ...
350 // ...
351 // ...
352 // ...
353 // ...
354 // ...
355 // ...
356 // ...
357 // ...
358 // ...
359 // ...
360 // ...
361 // ...
362 // ...
363 // ...
364 // ...
365 // ...
366 // ...
367 // ...
368 // ...
369 // ...
370 // ...
371 // ...
372 // ...
373 // ...
374 // ...
375 // ...
376 // ...
377 // ...
378 // ...
379 // ...
380 // ...
381 // ...
382 // ...
383 // ...
384 // ...
385 // ...
386 // ...
387 // ...
388 // ...
389 // ...
390 // ...
391 // ...
392 // ...
393 // ...
394 // ...
395 // ...
396 // ...
397 // ...
398 // ...
399 // ...
400 // ...
401 // ...
402 // ...
403 // ...
404 // ...
405 // ...
406 // ...
407 // ...
408 // ...
409 // ...
410 // ...
411 // ...
412 // ...
413 // ...
414 // ...
415 // ...
416 // ...
417 // ...
418 // ...
419 // ...
420 // ...
421 // ...
422 // ...
423 // ...
424 // ...
425 // ...
426 // ...
427 // ...
428 // ...
429 // ...
430 // ...
431 // ...
432 // ...
433 // ...
434 // ...
435 // ...
436 // ...
437 // ...
438 // ...
439 // ...
440 // ...
441 // ...
442 // ...
443 // ...
444 // ...
445 // ...
446 // ...
447 // ...
448 // ...
449 // ...
450 // ...
451 // ...
452 // ...
453 // ...
454 // ...
455 // ...
456 // ...
457 // ...
458 // ...
459 // ...
460 // ...
461 // ...
462 // ...
463 // ...
464 // ...
465 // ...
466 // ...
467 // ...
468 // ...
469 // ...
470 // ...
471 // ...
472 // ...
473 // ...
474 // ...
475 // ...
476 // ...
477 // ...
478 // ...
479 // ...
480 // ...
481 // ...
482 // ...
483 // ...
484 // ...
485 // ...
486 // ...
487 // ...
488 // ...
489 // ...
490 // ...
491 // ...
492 // ...
493 // ...
494 // ...
495 // ...
496 // ...
497 // ...
498 // ...
499 // ...
500 // ...
501 // ...
502 // ...
503 // ...
504 // ...
505 // ...
506 // ...
507 // ...
508 // ...
509 // ...
510 // ...
511 // ...
512 // ...
513 // ...
514 // ...
515 // ...
516 // ...
517 // ...
518 // ...
519 // ...
520 // ...
521 // ...
522 // ...
523 // ...
524 // ...
525 // ...
526 // ...
527 // ...
528 // ...
529 // ...
530 // ...
531 // ...
532 // ...
533 // ...
534 // ...
535 // ...
536 // ...
537 // ...
538 // ...
539 // ...
540 // ...
541 // ...
542 // ...
543 // ...
544 // ...
545 // ...
546 // ...
547 // ...
548 // ...
549 // ...
550 // ...
551 // ...
552 // ...
553 // ...
554 // ...
555 // ...
556 // ...
557 // ...
558 // ...
559 // ...
560 // ...
561 // ...
562 // ...
563 // ...
564 // ...
565 // ...
566 // ...
567 // ...
568 // ...
569 // ...
570 // ...
571 // ...
572 // ...
573 // ...
574 // ...
575 // ...
576 // ...
577 // ...
578 // ...
579 // ...
580 // ...
581 // ...
582 // ...
583 // ...
584 // ...
585 // ...
586 // ...
587 // ...
588 // ...
589 // ...
590 // ...
591 // ...
592 // ...
593 // ...
594 // ...
595 // ...
596 // ...
597 // ...
598 // ...
599 // ...
600 // ...
601 // ...
602 // ...
603 // ...
604 // ...
605 // ...
606 // ...
607 // ...
608 // ...
609 // ...
610 // ...
611 // ...
612 // ...
613 // ...
614 // ...
615 // ...
616 // ...
617 // ...
618 // ...
619 // ...
620 // ...
621 // ...
622 // ...
623 // ...
624 // ...
625 // ...
626 // ...
627 // ...
628 // ...
629 // ...
630 // ...
631 // ...
632 // ...
633 // ...
634 // ...
635 // ...
636 // ...
637 // ...
638 // ...
639 // ...
640 // ...
641 // ...
642 // ...
643 // ...
644 // ...
645 // ...
646 // ...
647 // ...
648 // ...
649 // ...
650 // ...
651 // ...
652 // ...
653 // ...
654 // ...
655 // ...
656 // ...
657 // ...
658 // ...
659 // ...
660 // ...
661 // ...
662 // ...
663 // ...
664 // ...
665 // ...
666 // ...
667 // ...
668 // ...
669 // ...
670 // ...
671 // ...
672 // ...
673 // ...
674 // ...
675 // ...
676 // ...
677 // ...
678 // ...
679 // ...
680 // ...
681 // ...
682 // ...
683 // ...
684 // ...
685 // ...
686 // ...
687 // ...
688 // ...
689 // ...
690 // ...
691 // ...
692 // ...
693 // ...
694 // ...
695 // ...
696 // ...
697 // ...
698 // ...
699 // ...
700 // ...
701 // ...
702 // ...
703 // ...
704 // ...
705 // ...
706 // ...
707 // ...
708 // ...
709 // ...
710 // ...
711 // ...
712 // ...
713 // ...
714 // ...
715 // ...
716 // ...
717 // ...
718 // ...
719 // ...
720 // ...
721 // ...
722 // ...
723 // ...
724 // ...
725 // ...
726 // ...
727 // ...
728 // ...
729 // ...
730 // ...
731 // ...
732 // ...
733 // ...
734 // ...
735 // ...
736 // ...
737 // ...
738 // ...
739 // ...
740 // ...
741 // ...
742 // ...
743 // ...
744 // ...
745 // ...
746 // ...
747 // ...
748 // ...
749 // ...
750 // ...
751 // ...
752 // ...
753 // ...
754 // ...
755 // ...
756 // ...
757 // ...
758 // ...
759 // ...
760 // ...
761 // ...
762 // ...
763 // ...
764 // ...
765 // ...
766 // ...
767 // ...
768 // ...
769 // ...
770 // ...
771 // ...
772 // ...
773 // ...
774 // ...
775 // ...
776 // ...
777 // ...
778 // ...
779 // ...
780 // ...
781 // ...
782 // ...
783 // ...
784 // ...
785 // ...
786 // ...
787 // ...
788 // ...
789 // ...
790 // ...
791 // ...
792 // ...
793 // ...
794 // ...
795 // ...
796 // ...
797 // ...
798 // ...
799 // ...
800 // ...
801 // ...
802 // ...
803 // ...
804 // ...
805 // ...
806 // ...
807 // ...
808 // ...
809 // ...
810 // ...
811 // ...
812 // ...
813 // ...
814 // ...
815 // ...
816 // ...
817 // ...
818 // ...
819 // ...
820 // ...
821 // ...
822 // ...
823 // ...
824 // ...
825 // ...
826 // ...
827 // ...
828 // ...
829 // ...
830 // ...
831 // ...
832 // ...
833 // ...
834 // ...
835 // ...
836 // ...
837 // ...
838 // ...
839 // ...
840 // ...
841 // ...
842 // ...
843 // ...
844 // ...
845 // ...
846 // ...
847 // ...
848 // ...
849 // ...
850 // ...
851 // ...
852 // ...
853 // ...
854 // ...
855 // ...
856 // ...
857 // ...
858 // ...
859 // ...
860 // ...
861 // ...
862 // ...
863 // ...
864 // ...
865 // ...
866 // ...
867 // ...
868 // ...
869 // ...
870 // ...
871 // ...
872 // ...
873 // ...
874 // ...
875 // ...
876 // ...
877 // ...
878 // ...
879 // ...
880 // ...
881 // ...
882 // ...
883 // ...
884 // ...
885 // ...
886 // ...
887 // ...
888 // ...
889 // ...
890 // ...
891 // ...
892 // ...
893 // ...
894 // ...
895 // ...
896 // ...
897 // ...
898 // ...
899 // ...
900 // ...
901 // ...
902 // ...
903 // ...
904 // ...
905 // ...
906 // ...
907 // ...
908 // ...
909 // ...
910 // ...
911 // ...
912 // ...
913 // ...
914 // ...
915 // ...
916 // ...
917 // ...
918 // ...
919 // ...
920 // ...
921 // ...
922 // ...
923 // ...
924 // ...
925 // ...
926 // ...
927 // ...
928 // ...
929 // ...
930 // ...
931 // ...
932 // ...
933 // ...
934 // ...
935 // ...
936 // ...
937 // ...
938 // ...
939 // ...
940 // ...
941 // ...
942 // ...
943 // ...
944 // ...
945 // ...
946 // ...
947 // ...
948 // ...
949 // ...
950 // ...
951 // ...
952 // ...
953 // ...
954 // ...
955 // ...
956 // ...
957 // ...
958 // ...
959 // ...
960 // ...
961 // ...
962 // ...
963 // ...
964 // ...
965 // ...
966 // ...
967 // ...
968 // ...
969 // ...
970 // ...
971 // ...
972 // ...
973 // ...
974 // ...
975 // ...
976 // ...
977 // ...
978 // ...
979 // ...
980 // ...
981 // ...
982 // ...
983 // ...
984 // ...
985 // ...
986 // ...
987 // ...
988 // ...
989 // ...
990 // ...
991 // ...
992 // ...
993 // ...
994 // ...
995 // ...
996 // ...
997 // ...
998 // ...
999 // ...
1000 // ...

```

```

Number of resources: 4
Number of processes: 3

Enter Maximum instances for each resource:
A's max : 6
B's max : 5
C's max : 7
D's max : 6

Enter Allocated instances of resource for each process:
P0's allocated : 1 2 2 1
P1's allocated : 1 0 3 3
P2's allocated : 1 2 1 0

Enter Maximum instances of resource required for each process:
P0's needed : 2 1 0 1
P1's needed : 0 2 0 1
P2's needed : 0 1 4 0

```

```

22 fun compute() {
23     var safeSequence: MutableList<Process> = mutableListOf()
24     var done = false;
25     var count = 0
26     while (!done) {
27         var i = 0
28         while (i < procList.size && procList.size != 0) {
29             // if (procList.size == 0 || count >= 2 * nProcess) {
30             //     done = true
31             //     break
32             // }
33             var sufficient = true
34             for (j in 0 until resList.size) {
35                 if (procList[i].need[j] - procList[i].taken[j] > resList[j].available) {
36                     sufficient = false
37                     break
38                 }
39             }
40             if (sufficient) {
41                 for (k in 0 until resList.size) {
42                     resList[k].available += procList[i].taken[k]
43                 }
44                 println("Executing process ${procList[i].name}")
45                 availableResource()
46                 safeSequence.add(procList[i])
47                 procList.removeAt(i)
48                 i = 0
49             }
50             i++
51             count++
52         }
53         if (procList.size == 0 || count >= 2 * nProcess) {
54             done = true
55             break
56         }
57     }
58 }
59 if (count >= 2 * nProcess) {
60     println("No possible solution")
61     println("Deadlock occur")
62 }
63 println("Safe Sequence:")
64 print("< ")
65 for (i in safeSequence) {
66     print("${i.name} ")
67 }
68 print("> ")
69 }

```

1 by 1 the program will try and look for a safe state

BankersAlgo.kt ✕ Process.kt Resource.kt

If the safe state is found for all the process
demanding the resource

```

27 var i = 0
28 while (i < procList.size && procList.size != 0) {
29 //     if (procList.size == 0 || count >= 2 * nProcess) {
30 //         done = true
31 //         break
32 //     }
33     var sufficient = true
34     for (j in 0 until resList.size) {
35         if (procList[i].need[j] - procList[i].taken[j] > resList[j].available) {
36             sufficient = false
37             break
38         }
39     }
40     if (sufficient) {
41         for (k in 0 until resList.size) {
42             resList[k].available += procList[i].taken[k]
43         }
44         println("Executing process ${procList[i].name}")
45         availableResource()
46         safeSequence.add(procList[i])
47         procList.removeAt(i)
48         i = 0
49     }
50
51     i++
52     count++
53 }
54 if (procList.size == 0 || count >= 2 * nProcess) {
55     done = true
56 //     break
57 }
58 }
59 if (count >= 2 * nProcess) {
60     println("No possible solution")
61     println("Deadlock occur")
62 }
63 println("Safe Sequence:")
64 print("< ")
65 for (i in safeSequence) {
66     print("${i.name} ")
67 }
68 print("> ")
69

```

The program execute that state and release
the resource


```

34     for (j in 0 until resList.size) {
35         if (procList[i].need[j] - procList[i].taken[j] > resList[j].available) {
36             sufficient = false
37             break
38         }
39     }
40     if (sufficient) {
41         for (k in 0 until resList.size) {
42             resList[k].available += procList[i].taken[k]
43         }
44         println("Executing process ${procList[i].name}")
45         availableResource()
46         safeSequence.add(procList[i])
47         procList.removeAt(i)
48         i = 0
49     }
50
51     i++
52     count++
53 }
54 if (procList.size == 0 || count >= 2 * nProcess) {
55     done = true
56     // break
57 }
58 }
59 if (count >= 2 * nProcess) {
60     println("No possible solution")
61     println("Deadlock occur")

```

```

34     for (j in 0 until resList.size) {
35         if (procList[i].need[j] - procList[i].taken[j] > resList[j].available) {
36             sufficient = false
37             break
38         }
39     }
40
41
42
43
44
45
46
47
48
49
50
51     i++
52     count++
53 }
54 if (procList.size == 0 || count >= 2 * nProcess) {
55     done = true
56     break
57 }
58 }
59 if (count >= 2 * nProcess) {
60     println("No possible solution")
61     println("Deadlock occur")

```

```

34     for (j in 0 until resList.size) {
35         if (procList[i].need[j] - procList[i].taken[j] > resList[j].available) {
36             sufficient = false
37             break
38         }
39     }
40
41
42
43
44
45
46
47
48
49
50
51     i++
52     count++
53 }
54 if (procList.size == 0 || count >= 2 * nProcess) {
55     done = true
56     break
57 }
58 }
59 if (count >= 2 * nProcess) {
60     println("No possible solution")
61     println("Deadlock occur")

```

```

27 var i = 0
28 while (i < procList.size && procList.size != 0) {
29 //     if (procList.size == 0 || count >= 2 * nProcess) {
30 //         done = true
31 //         break
32 //     }
33 var sufficient = true
34 for (j in 0 until resList.size) {
35     if (procList[i].need[j] - procList[i].taken[j] > resList[j].available) {
36         sufficient = false
37         break
38     }
39 }
40 if (sufficient) {
41     for (k in 0 until resList.size) {
42         resList[k].available += procList[i].taken[k]
43     }
44     println("Executing process ${procList[i].name}")
45     availableResource()
46     safeSequence.add(procList[i])
47     procList.removeAt(i)
48     i = 0
49 }
50
51 i++
52 count++
53 }
54 if (procList.size == 0 || count >= 2 * nProcess) {
55     done = true
56 //     break
57 }
58 }
59 if (count >= 2 * nProcess) {
60     println("No possible solution")
61     println("Deadlock occur")
62 }
63 println("Safe Sequence:")
64 print("< ")
65 for (i in safeSequence) {
66     print("${i.name} ")
67 }
68 print("> ")
69

```

Other wise if safe state is not found

```

39     }
40     if (sufficient) {
41         for (k in 0 until resList.size) {
42             resList[k].available += procList[i].taken[k]
43         }
44         println("Executing process ${procList[i].name}")
45         availableResource()
46         safeSequence.add(procList[i])
47         procList.removeAt(i)
48         i = 0
49     }
50
51     i++
52     count++
53 }
54 if (procList.size == 0 || count >= 2 * nProcess) {
55     done = true
56     // break
57 }
58 }
59 if (count >= 2 * nProcess) {
60     println("No possible solution")
61     println("Deadlock occur")
62 }
63 println("Safe Sequence:")
64 print("< ")
65 for (i in safeSequence) {
66     print("${i.name} ")
67 }
68 print("> ")
69

```

```

Enter Maximum instances for each resource:
A's max : 6
B's max : 5
C's max : 7
D's max : 6

Enter Allocated instances of resource for each process:
P0's allocated : 1 2 2 1
P1's allocated : 1 0 3 3
P2's allocated : 1 2 1 0

Enter Maximum instances of resource required for each process:
P0's needed : 2 1 0 1
P1's needed : 0 2 0 1
P2's needed : 0 1 4 0

```

It states the result as deadlock occur


```

BankersAlgo.kt x Process.kt Re
22 fun compute() {
23
24
25     1 package bankers
26     2
27     3 var resList: MutableList<Resource> = mutableListOf()
28     4 var procList: MutableList<Process> = mutableListOf()
29     5 var sum: MutableList<Int> = mutableListOf()
30     6 var nProcess = 0
31     7
32     8 fun main(args: Array<String>) {
33         9 init()
34         10 compute()
35     11 }
36
37     13 fun availableResource() {
38         14 print("Available Resource: ")
39         15 for (i in 0 until resList.size) {
40             16 print("${resList[i].available} ")
41             17 }
42             18 println()
43             19 println()
44             20 }
45
46     21
47     22 procList.removeAt(i)
48     23 i = 0
49     24 }
50
51     25 i++
52     26 count++
53     27 }
54     28 if (procList.size == 0)
55     29 done = true
56     30 break
57     31 }
58
59     32 if (count >= 2 * nProcess) {
60         33 println("No possible safe sequence")
61         34 println("Deadlock occurs")
62     35 }
63     36 println("Safe Sequence:")
64     37 print("< ")
65     38 for (i in safeSequence) {
66         39 print("${i.name} ")
67     40 }
68     41 print("> ")
69
70     42
71     43
72     44
73     45
74     46
75     47
76     48
77     49
78     50
79     51
80     52
81     53
82     54
83     55
84     56
85     57
86     58
87     59
88     60
89     61
90     62
91     63
92     64
93     65
94     66
95     67
96     68
97     69
98     69
99     70
100    71
101    72
102    73
103    74
104    75
105    76
106    77
107    78
108    79
109    80
110    81
111    82
112    83
113    84
114    85
115    86
116    87
117    88
118    89
119    90
120    91
121    92
122    93
123    94
124    95
125    96
126    97
127    98
128    99
129    100
130    101
131    102
132    103
133    104
134    105
135    106
136    107
137    108
138    109
139    110
140    111
141    112
142    113
143    114
144    115
145    116
146    117
147    118
148    119
149    120
150    121
151    122
152    123
153    124
154    125
155    126
156    127
157    128
158    129
159    130
160    131
161    132
162    133
163    134
164    135
165    136
166    137
167    138
168    139
169    140
170    141
171    142
172    143
173    144
174    145
175    146
176    147
177    148
178    149
179    150
180    151
181    152
182    153
183    154
184    155
185    156
186    157
187    158
188    159
189    160
190    161
191    162
192    163
193    164
194    165
195    166
196    167
197    168
198    169
199    170
200    171
201    172
202    173
203    174
204    175
205    176
206    177
207    178
208    179
209    180
210    181
211    182
212    183
213    184
214    185
215    186
216    187
217    188
218    189
219    190
220    191
221    192
222    193
223    194
224    195
225    196
226    197
227    198
228    199
229    200
230    201
231    202
232    203
233    204
234    205
235    206
236    207
237    208
238    209
239    210
240    211
241    212
242    213
243    214
244    215
245    216
246    217
247    218
248    219
249    220
250    221
251    222
252    223
253    224
254    225
255    226
256    227
257    228
258    229
259    230
260    231
261    232
262    233
263    234
264    235
265    236
266    237
267    238
268    239
269    240
270    241
271    242
272    243
273    244
274    245
275    246
276    247
277    248
278    249
279    250
280    251
281    252
282    253
283    254
284    255
285    256
286    257
287    258
288    259
289    260
290    261
291    262
292    263
293    264
294    265
295    266
296    267
297    268
298    269
299    270
300    271
301    272
302    273
303    274
304    275
305    276
306    277
307    278
308    279
309    280
310    281
311    282
312    283
313    284
314    285
315    286
316    287
317    288
318    289
319    290
320    291
321    292
322    293
323    294
324    295
325    296
326    297
327    298
328    299
329    300
330    301
331    302
332    303
333    304
334    305
335    306
336    307
337    308
338    309
339    310
340    311
341    312
342    313
343    314
344    315
345    316
346    317
347    318
348    319
349    320
350    321
351    322
352    323
353    324
354    325
355    326
356    327
357    328
358    329
359    330
360    331
361    332
362    333
363    334
364    335
365    336
366    337
367    338
368    339
369    340
370    341
371    342
372    343
373    344
374    345
375    346
376    347
377    348
378    349
379    350
380    351
381    352
382    353
383    354
384    355
385    356
386    357
387    358
388    359
389    360
390    361
391    362
392    363
393    364
394    365
395    366
396    367
397    368
398    369
399    370
400    371
401    372
402    373
403    374
404    375
405    376
406    377
407    378
408    379
409    380
410    381
411    382
412    383
413    384
414    385
415    386
416    387
417    388
418    389
419    390
420    391
421    392
422    393
423    394
424    395
425    396
426    397
427    398
428    399
429    400
430    401
431    402
432    403
433    404
434    405
435    406
436    407
437    408
438    409
439    410
440    411
441    412
442    413
443    414
444    415
445    416
446    417
447    418
448    419
449    420
450    421
451    422
452    423
453    424
454    425
455    426
456    427
457    428
458    429
459    430
460    431
461    432
462    433
463    434
464    435
465    436
466    437
467    438
468    439
469    440
470    441
471    442
472    443
473    444
474    445
475    446
476    447
477    448
478    449
479    450
480    451
481    452
482    453
483    454
484    455
485    456
486    457
487    458
488    459
489    460
490    461
491    462
492    463
493    464
494    465
495    466
496    467
497    468
498    469
499    470
500    471
501    472
502    473
503    474
504    475
505    476
506    477
507    478
508    479
509    480
510    481
511    482
512    483
513    484
514    485
515    486
516    487
517    488
518    489
519    490
520    491
521    492
522    493
523    494
524    495
525    496
526    497
527    498
528    499
529    500
530    501
531    502
532    503
533    504
534    505
535    506
536    507
537    508
538    509
539    510
540    511
541    512
542    513
543    514
544    515
545    516
546    517
547    518
548    519
549    520
550    521
551    522
552    523
553    524
554    525
555    526
556    527
557    528
558    529
559    530
560    531
561    532
562    533
563    534
564    535
565    536
566    537
567    538
568    539
569    540
570    541
571    542
572    543
573    544
574    545
575    546
576    547
577    548
578    549
579    550
580    551
581    552
582    553
583    554
584    555
585    556
586    557
587    558
588    559
589    560
590    561
591    562
592    563
593    564
594    565
595    566
596    567
597    568
598    569
599    570
600    571
601    572
602    573
603    574
604    575
605    576
606    577
607    578
608    579
609    580
610    581
611    582
612    583
613    584
614    585
615    586
616    587
617    588
618    589
619    590
620    591
621    592
622    593
623    594
624    595
625    596
626    597
627    598
628    599
629    600
630    601
631    602
632    603
633    604
634    605
635    606
636    607
637    608
638    609
639    610
640    611
641    612
642    613
643    614
644    615
645    616
646    617
647    618
648    619
649    620
650    621
651    622
652    623
653    624
654    625
655    626
656    627
657    628
658    629
659    630
660    631
661    632
662    633
663    634
664    635
665    636
666    637
667    638
668    639
669    640
670    641
671    642
672    643
673    644
674    645
675    646
676    647
677    648
678    649
679    650
680    651
681    652
682    653
683    654
684    655
685    656
686    657
687    658
688    659
689    660
690    661
691    662
692    663
693    664
694    665
695    666
696    667
697    668
698    669
699    670
700    671
701    672
702    673
703    674
704    675
705    676
706    677
707    678
708    679
709    680
710    681
711    682
712    683
713    684
714    685
715    686
716    687
717    688
718    689
719    690
720    691
721    692
722    693
723    694
724    695
725    696
726    697
727    698
728    699
729    700
730    701
731    702
732    703
733    704
734    705
735    706
736    707
737    708
738    709
739    710
740    711
741    712
742    713
743    714
744    715
745    716
746    717
747    718
748    719
749    720
750    721
751    722
752    723
753    724
754    725
755    726
756    727
757    728
758    729
759    730
760    731
761    732
762    733
763    734
764    735
765    736
766    737
767    738
768    739
769    740
770    741
771    742
772    743
773    744
774    745
775    746
776    747
777    748
778    749
779    750
780    751
781    752
782    753
783    754
784    755
785    756
786    757
787    758
788    759
789    760
790    761
791    762
792    763
793    764
794    765
795    766
796    767
797    768
798    769
799    770
800    771
801    772
802    773
803    774
804    775
805    776
806    777
807    778
808    779
809    780
810    781
811    782
812    783
813    784
814    785
815    786
816    787
817    788
818    789
819    790
820    791
821    792
822    793
823    794
824    795
825    796
826    797
827    798
828    799
829    800
830    801
831    802
832    803
833    804
834    805
835    806
836    807
837    808
838    809
839    810
840    811
841    812
842    813
843    814
844    815
845    816
846    817
847    818
848    819
849    820
850    821
851    822
852    823
853    824
854    825
855    826
856    827
857    828
858    829
859    830
860    831
861    832
862    833
863    834
864    835
865    836
866    837
867    838
868    839
869    840
870    841
871    842
872    843
873    844
874    845
875    846
876    847
877    848
878    849
879    850
880    851
881    852
882    853
883    854
884    855
885    856
886    857
887    858
888    859
889    860
890    861
891    862
892    863
893    864
894    865
895    866
896    867
897    868
898    869
899    870
900    871
901    872
902    873
903    874
904    875
905    876
906    877
907    878
908    879
909    880
910    881
911    882
912    883
913    884
914    885
915    886
916    887
917    888
918    889
919    890
920    891
921    892
922    893
923    894
924    895
925    896
926    897
927    898
928    899
929    900
930    901
931    902
932    903
933    904
934    905
935    906
936    907
937    908
938    909
939    910
940    911
941    912
942    913
943    914
944    915
945    916
946    917
947    918
948    919
949    920
950    921
951    922
952    923
953    924
954    925
955    926
956    927
957    928
958    929
959    930
960    931
961    932
962    933
963    934
964    935
965    936
966    937
967    938
968    939
969    940
970    941
971    942
972    943
973    944
974    945
975    946
976    947
977    948
978    949
979    950
980    951
981    952
982    953
983    954
984    955
985    956
986    957
987    958
988    959
989    960
990    961
991    962
992    963
993    964
994    965
995    966
996    967
997    968
998    969
999    970
1000   971
1001   972
1002   973
1003   974
1004   975
1005   976
1006   977
1007   978
1008   979
1009   980
1010   981
1011   982
1012   983
1013   984
1014   985
1015   986
1016   987
1017   988
1018   989
1019   990
1020   991
1021   992
1022   993
1023   994
1024   995
1025   996
1026   997
1027   998
1028   999
1029   1000
1030   1001
1031   1002
1032   1003
1033   1004
1034   1005
1035   1006
1036   1007
1037   1008
1038   1009
1039   1010
1040   1011
1041   1012
1042   1013
1043   1014
1044   1015
1045   1016
1046   1017
1047   1018
1048   1019
1049   1020
1050   1021
1051   1022
1052   1023
1053   1024
1054   1025
1055   1026
1056   1027
1057   1028
1058   1029
1059   1030
1060   1031
1061   1032
1062   1033
1063   1034
1064   1035
1065   1036
1066   1037
1067   1038
1068   1039
1069   1040
1070   1041
1071   1042
1072   1043
1073   1044
1074   1045
1075   1046
1076   1047
1077   1048
1078   1049
1079   1050
1080   1051
1081   1052
1082   1053
1083   1054
1084   1055
1085   1056
1086   1057
1087   1058
1088   1059
1089   1060
1090   1061
1091   1062
1092   1063
1093   1064
1094   1065
1095   1066
1096   1067
1097   1068
1098   1069
1099   1070
1100   1071
1101   1072
1102   1073
1103   1074
1104   1075
1105   1076
1106   1077
1107   1078
1108   1079
1109   1080
1110   1081
1111   1082
1112   1083
1113   1084
1114   1085
1115   1086
1116   1087
1117   1088
1118   1089
1119   1090
1120   1091
1121   1092
1122   1093
1123   1094
1124   1095
1125   1096
1126   1097
1127   1098
1128   1099
1129   1100
1130   1101
1131   1102
1132   1103
1133   1104
1134   1105
1135   1106
1136   1107
1137   1108
1138   1109
1139   1110
1140   1111
1141   1112
1142   1113
1143   1114
1144   1115
1145   1116
1146   1117
1147   1118
1148   1119
1149   1120
1150   1121
1151   1122
1152   1123
1153   1124
1154   1125
1155   1126
1156   1127
1157   1128
1158   1129
1159   1130
1160   1131
1161   1132
1162   1133
1163   1134
1164   1135
1165   1136
1166   1137
1167   1138
1168   1139
1169   1140
1170   1141
1171   1142
1172   1143
1173   1144
1174   1145
1175   1146
1176   1147
1177   1148
1178   1149
1179   1150
1180   1151
1181   1152
1182   1153
1183   1154
1184   1155
1185   1156
1186   1157
1187   1158
1188   1159
1189   1160
1190   1161
1191   1162
1192   1163
1193   1164
1194   1165
1195   1166
1196   1167
1197   1168
1198   1169
1199   1170
1200   1171
1201   1172
1202   1173
1203   1174
1204   1175
1205   1176
1206   1177
1207   1178
1208   1179
1209   1180
1210   1181
1211   1182
1212   1183
1213   1184
1214   1185
1215   1186
1216   1187
1217   1188
1218   1189
1219   1190
1220   1191
1221   1192
1222   1193
1223   1194
1224   1195
1225   1196
1226   1197
1227   1198
1228   1199
1229   1200
1230   1201
1231   1202
1232   1203
1233   1204
1234   1205
1235   1206
1236   1207
1237   1208
1238   1209
1239   1210
1240   1211
1241   1212
1242   1213
1243   1214
1244   1215
1245   1216
1246   1217
1247   1218
1248   1219
1249   1220
1250   1221
1251   1222
1252   1223
1253   1224
1254   1225
1255   1226
1256   1227
1257   1228
1258   1229
1259   1230
1260   1231
1261   1232
1262   1233
1263   1234
1264   1235
1265   1236
1266   1237
1267   1238
1268   1239
1269   1240
1270   1241
1271   1242
1272   1243
1273   1244
1274   1245
1275   1246
1276   1247
1277   1248
1278   1249
1279   1250
1280   1251
1281   1252
1282   1253
1283   1254
1284   1255
1285   1256
1286   1257
1287   1258
1288   1259
1289   1260
1290   1261
1291   1262
1292   1263
1293   1264
1294   1265
1295   1266
1296   1267
1297   1268
1298   1269
1299   1270
1300   1271
1301   1272
1302   1273
1303   1274
1304   1275
1305   1276
1306   1277
1307   1278
1308   1279
1309   1280
1310   1281
1311   1282
1312   1283
1313   1284
1314   1285
1315   1286
1316   1287
1317   1288
1318   1289
1319   1290
1320   1291
1321   1292
1322   1293
1323   1294
1324   1295
1325   1296
1326   1297
1327   12
```

Deadlock Avoidance

Banker's Algorithm

Happy Coding!

THANK YOU