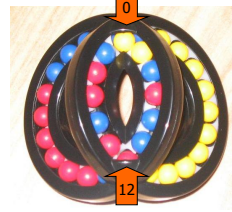


Part II

Puzzle solving



(Redundant) representation of a configuration (start state)



- State comprises of:
 - Two 18 bead wheels
 - $2 \times 18 - 2 = 34$ beads (total)
 - 12 yellow beads (centre)
 - 11 red beads
 - 11 blue beads

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
w1	anti-clock	y	b	b	b	r	r	r	r	r	r	r	b	r	b	b	r	b	y
w2	clockwise	y	b	y	y	y	y	y	y	y	y	y	y	r	r	b	r	r	b

Final configuration (final state)



		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
w1	anti-clock	y	b	b	b	b	b	b	b	b	b	b	y	y	y	y	y	y	y
w2	clockwise	y	r	r	r	r	r	r	r	r	r	r	r	y	y	y	y	y	y

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
w1	anti-clock	y	b	b	b	r	r	r	r	r	r	r	b	r	b	b	r	b	y
w2	clockwise	y	b	y	y	y	y	y	y	y	y	y	y	r	r	b	r	r	b



Shifting w1 clockwise by 1

Before (above)
After (below)

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
w1	anti-clock	b	b	b	r	r	r	r	r	r	r	b	r	b	b	r	b	y	y
w2	clockwise	b	b	y	y	y	y	y	y	y	y	y	y	b	r	b	r	r	b

Generic configurations

Original and specific puzzles:

- 18 11/7/4 bead wheels
- 12 6/4/2 yellow beads
- 11 7/4/2 blue and red beads

Generic puzzle:

- w bead wheels
- y yellow beads (y even because of vertical symmetry)
- $b = r = (2w - 2 - y)/2$ blue and red beads

	0													r+1					w-1
w1	y	b	b	b	b	b	b	b	b	b	b	b	y	y	y	y	y	y	y
w2	y	r	r	r	r	r	r	r	r	r	r	r	y	y	y	y	y	y	y

nextConfigs method (1 of 2)

```
private LinkedList<String> nextConfigs(String config)
{ // 1
    String[] wheels = new String[2];
    wheels[0] = config.substring(0, w);
    wheels[1] = config.substring(w);
    LinkedList<String> result = new LinkedList<String>();
    for (int i = 0; i < 2; i++) /* wheel index */
    { // 2
        for (int j = 1; j < w; j++) /* rotation */
        { // 3
            char[] shiftedWheel = new char[w];
            for (int k = 0; k < w; k++) /* location */
                shiftedWheel[k] = wheels[i].charAt((k+j) % w);
            String nextConfig;
```

nextConfigs method (2 of 2)

```
String nextConfig;
if (i == 0)
    nextConfig = String.valueOf(shiftedWheel) +
        shiftedWheel[0] +
        wheels[1].substring(1, r + 1) +
        shiftedWheel[r + 1] +
        wheels[1].substring(r + 2);
else
    nextConfig = shiftedWheel[0] +
        wheels[0].substring(1, r + 1) +
        shiftedWheel[r + 1] +
        wheels[0].substring(r + 2) +
        String.valueOf(shiftedWheel);
if (!config.equals(nextConfig) && !result.contains(nextConfig))
    result.add(nextConfig);
} // 3
} // 2
return result;
} // 1
```

On the disequality and inclusion test

- Consider next configurations of rbrb ryyb:

left shift	Wheel 1	both	Wheel 2	both
1	brbr	brbr byyr	yybr	ybrr yybr
2	rbrb	rbrb ryyb	ybyr	ybyr ybyr
3	brbr	brbr byyr	bryy	bbry bryy

- Thus set of only 5 configurations
- $2(w-1)$ in worse-case as shown by rbrb ryyr:
 - bbrrbyyr, brbrbbyb, rrbryyb, ybbryyr, ybbryyr, rbbryyr

depthFirst method (1 of 2)

```
private LinkedList<String> depthFirst(String config, String dest, int depth)
{ // 1
    if (depth == 0) return null;
    else if (config.equals(dest))
    {
        LinkedList<String> route = new LinkedList<String>();
        route.add(config);
        return route;
    }
    else
    {
```

depthFirst method (2 of 2)

```
else
{
    LinkedList<String> result = nextConfigs(config);
    for (String nextConfig: result)
    {
        LinkedList<String>
            route = depthFirst(nextConfig, dest, depth - 1);
        if (route != null)
        {
            route.addFirst(config);
            return route;
        }
    }
    return null;
} // 1
```

iterativeDeepening method

```
public LinkedList<String> iterativeDeepening(String config, String dest)
{
    for (int depth = 1; true; depth++)
    {
        System.out.println(depth);
        LinkedList<String> route = depthFirst(config, dest, depth);
        if (route != null) return route;
    }
}
```

Sample runs

```
>java BeadFinder yrryybby ybbryyr
1
2
3
4
    W1 W2 W1
[yrryybby, ryyrrbbr, byybbrrb, ybbryyr]
>
>java BeadFinder ybbryyr yrryybby
1
2
3
4
    W1 W2 W1
[ybbryyr, byybbrrb, ryyrrbbr, yrryybby]
```

River crossing puzzles

- n missionaries and n cannibals are one side of a river;
- And so is a boat that holds c people
- Find the most time efficient way of moving everyone to the other side without leaving a group of missionaries on either side outnumbered by the cannibals



[Machine Intelligence, 3, 1968]

What is a configuration?

- Start configuration $\langle n, n, \text{true} \rangle$
 - n missionaries, n cannibals and 1 boat on the initial side
- End configurations $\langle 0, 0, \text{true} \rangle$ and $\langle 0, 0, \text{false} \rangle$
 - 0 missionaries and 0 cannibals on initial side
- Consider next configurations for $\langle 2, 2, \text{true} \rangle$ for the $n = 4$ and $c = 2$ instance:

■ $\langle 0, 2, \text{false} \rangle$	✓	cc ~~~ mmmccc
■ $\langle 1, 1, \text{false} \rangle$	✓	mc ~~~ mmmccc
■ $\langle 2, 0, \text{false} \rangle$	×	mm ~~~ mmmccc

Triple class (1 of 2)

```
public class Triple
{
    private int miss, cann;
    private boolean boat;

    Triple(int miss, int cann, boolean boat)
    {
        this.miss = miss;
        this.cann = cann;
        this.boat = boat;
    }

    public int getMiss()
    {
        return miss;
    }
    // other accessor methods
}
```

Triple class (2 of 2)

```
public boolean isValid(int n)
{
    if (miss == n) return true;           // case 1
    if (miss == 0) return true;          // case 2
    // if (miss >= cann && (n - miss) >= (n - cann)) return true;
    // if (miss >= cann && (-miss) >= (-cann)) return true;
    // if (miss >= cann && cann >= miss) return true;
    if (miss == cann) return true;        // case 3
    return false;
}

public String toString()
{
    return "(" + miss + ", " + cann + ", " + boat + ")";
}
}
```

nextConfig() (1 of 2)

```
public LinkedList<Triple> nextConfigs(Triple config)
{
    LinkedList<Triple> result = new LinkedList<Triple>();
    for (int moveMiss = 0; moveMiss <= c; moveMiss++)
        for (int moveCann = 0; moveCann <= c - moveMiss; moveCann++)
            if (config.getBoat())
            {
                int newMiss = config.getMiss() - moveMiss;
                int newCann = config.getCann() - moveCann;
                if (newMiss < 0);
                else if (newCann < 0);
                else
                {
                    Triple triple = new Triple(newMiss, newCann, false);
                    if (triple.isValid(n)) result.add(triple);
                }
            }
    }
    else
}
```

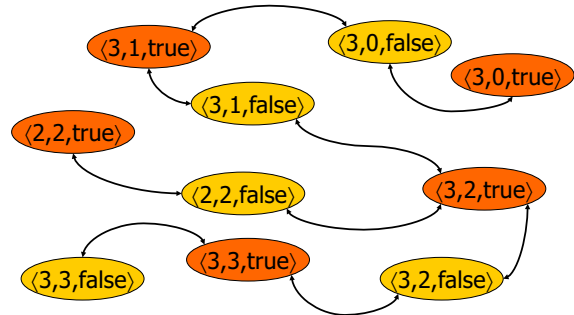
nextConfig() (2 of 2)

```
else
{
    int newMiss = config.getMiss() + moveMiss;
    int newCann = config.getCann() + moveCann;
    if (newMiss > n);
    else if (newCann > n);
    else
    {
        Triple triple = new Triple(newMiss, newCann, true);
        if (triple.isValid(n)) result.add(triple);
    }
}
return result;
} // 1
```

Sample runs with iterative deepening

```
>java BoatFinder 5 3
1
...
8
[(5, 5, true), (4, 4, false), (4, 4, true),
(3, 3, false), (3, 3, true), (0, 3, false),
(0, 3, true), (0, 0, false)]
>
>java BoatFinder 3 1
1
2
...
1043 ^C
```

Debugging n = 3 and c = 1



Counting configurations

- Consider $n = 3$ and initially ignore the boat:
- Cases 1 and 2 both give $4 = n+1$ configurations
- Case 3 gives $2 = n-1$ additional configurations
- Total of $2(2(n+1) + (n-1)) = 2(3n+1)$ configurations considering boat positions

mmm			ccc
mmm	c		cc
mmm	cc		c
mmm	ccc		
		mmm	ccc
	c	mmm	cc
	cc	mmm	c
	ccc	mmm	
		mmm	ccc
m	c	mm	cc
mm	cc	m	c
mmm	ccc		

depthLimitedIterativeDeepening method

```
public LinkedList<Triple> depthLimitedIterativeDeepening()
{
    for (int depth = 1; depth < 6*n+2; depth++)
    {
        System.out.println(depth);
        LinkedList<Triple> route = depthFirst(new Triple(n, n, true), depth);
        if (route != null) return route;
    }
    return null;
}
```

Sample runs with depth limited iterative deepening

```
>java BoatFinder 5 3
1
...
8
[(5, 5, true), (4, 4, false), (4, 4, true),
(3, 3, false), (3, 3, true), (0, 3, false),
(0, 3, true), (0, 0, false)]
>
>java BoatFinder 3 1
1
...
19
null
```

Generic concepts

	Route planning	Bead puzzle	Missionaries and cannibals	8-puzzle
state	town	configuration of 2 wheels	who is on what bank	?
operator	road graph	move graph	move graph	?
goal state(s)	destination town	unique target configuration	all people on other side of river	?
branching factor	~4	$2(w-1)$	~3	~4
solution	journey	series of wheel moves	series of boat and cargo moves	?