## Tries

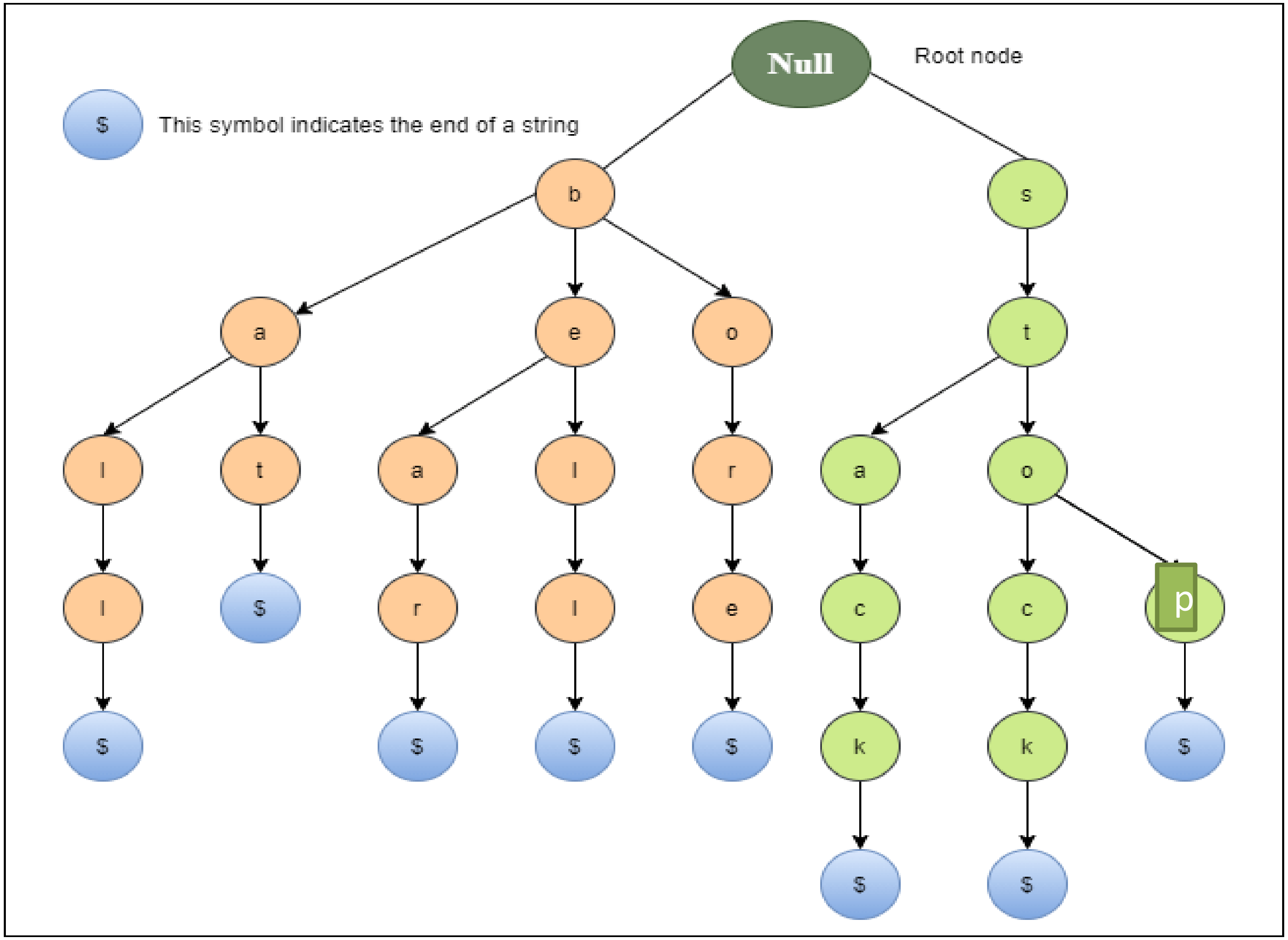
- The word "**Trie**" is an excerpt from the word "**retrieval**".
- Trie is a sorted tree-based data-structure that stores the set of strings

## Properties of the Trie for a set of the string:

- The root node of the trie always represents the null node.
- Each child of node is sorted alphabetically.
- Each node can have a maximum of **26** children (A to Z).
- Each node (except the root) can store one letter of the alphabet.

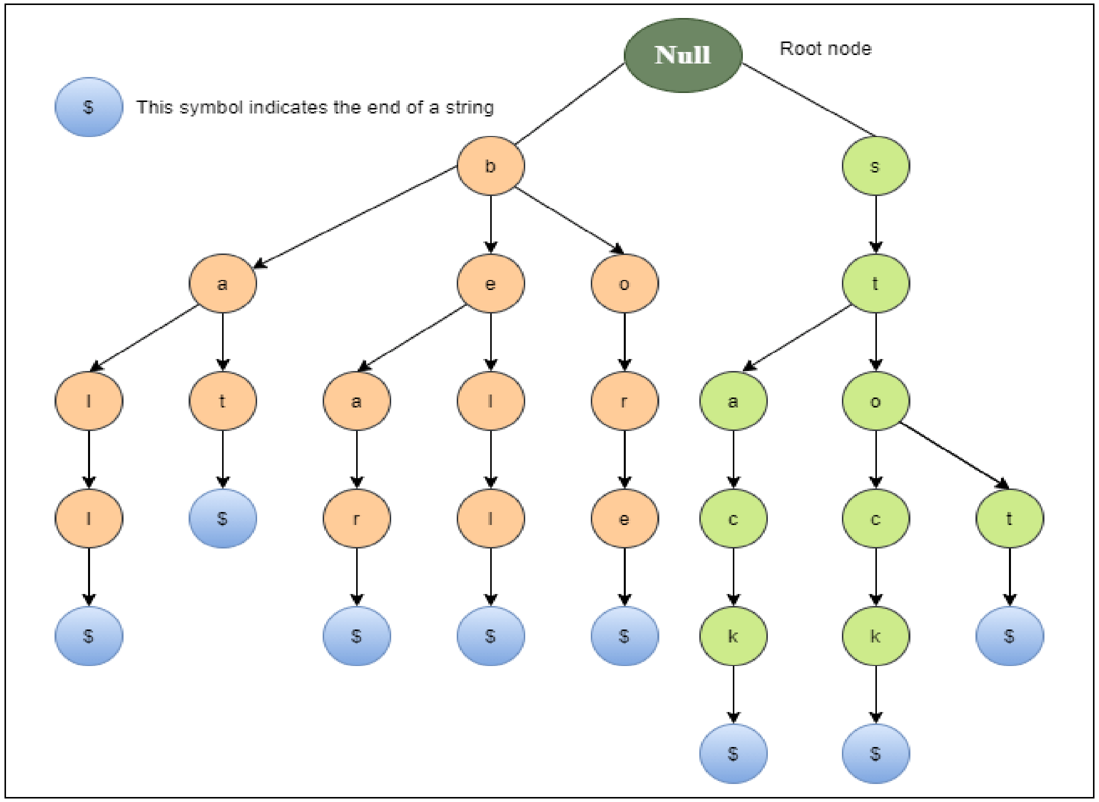
## Properties of the Trie for a set of the string:

A trie representation for the bell, bear, bore, bat, ball, stop, stock, and stack.



## Tries

- Every node of Trie consists of multiple branches. Each branch represents a possible character of keys.
- We need to mark the last node of every key as end of word node. A Trie node field *isEndOfWord* is used to distinguish the node as end of word node.



## Applications of Trie

**1. Spell Checker**

Spell checking is a three-step process.

1) First, look for that word in a dictionary,
2) generate possible suggestions, and
3) then sort the suggestion words with the desired word at the top.

## Applications of Trie

**1. Spell Checker**

- Trie is used to store the word in dictionaries.
- The spell checker can easily be applied in the most efficient way by searching for words on a data structure.
- Using trie not only makes it easy to see the word in the dictionary, but it is also simple to build an algorithm to include a collection of relevant words or suggestions.

## Applications of Trie

**2. Auto-complete**

- Auto-complete functionality is widely used on text editors, mobile applications, and the Internet.
- It provides a simple way to find an alternative word to complete the word for the following reasons.

- It provides an alphabetical filter of entries by the key of the node.
- We trace pointers only to get the node that represents the string entered by the user.
- As soon as you start typing, it tries to complete your input.

## Applications of Trie

**3. Browser history**

It is also used to complete the URL in the browser.

The browser keeps a history of the URLs of the websites you've visited.

## Advantage of Trie

1) With Trie, we can insert and find strings in O(L) time where L represent the length of a single word.
   - This is obviously faster than BST.
   - This is also faster than Hashing because of the ways it is implemented. We do not need to compute any hash function. No collision handling is required (like we do in open addressing and separate chaining)
2) Another advantage of Trie is, we can easily print all words in alphabetical order which is not easily possible with hashing.
3) We can efficiently do prefix search (or auto-complete) with Trie.

## Drawbacks of Trie

- **The main disadvantage of tries is that they need a lot of memory for storing the strings.**
- **For each node we have too many node pointers(equal to number of characters of the alphabet)**

## Suffix ?

- Suffixes are one or more letters added to the end of a base word to change its conjugation, word type, or other grammar properties like plurality.

Eg-

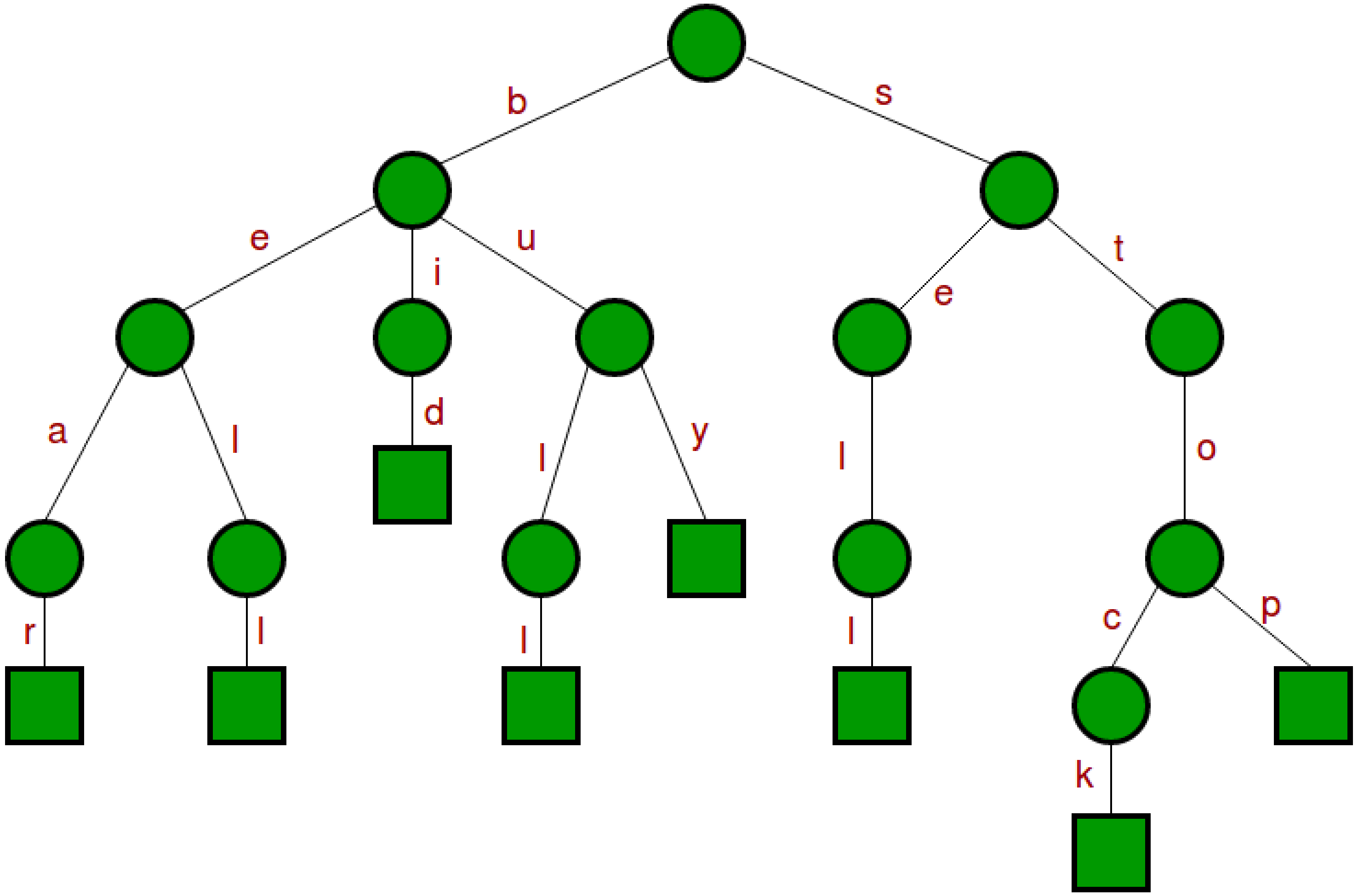
- you can add the suffix *-s* to the noun *strength* to make it plural (*strengths*) or
- add the suffix –*en* to turn it into a verb (*strengthen*)
- adding –*ment* to the verb *pay* to make the noun *payment*.

## Suffix Tree

- A Suffix Tree for a given text is a compressed trie for all suffixes of the given text.

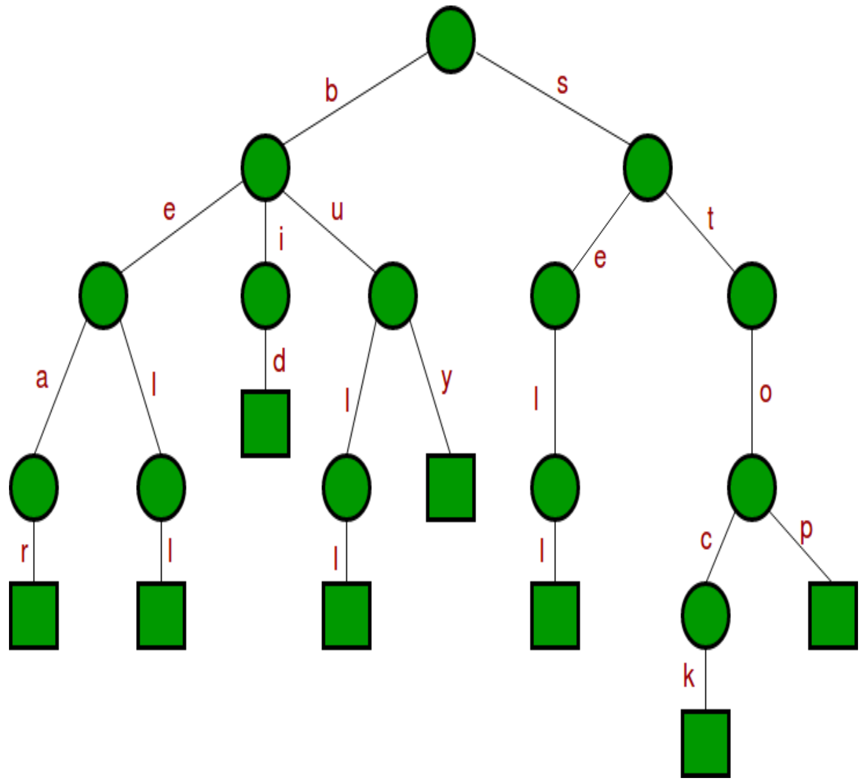
# Trie

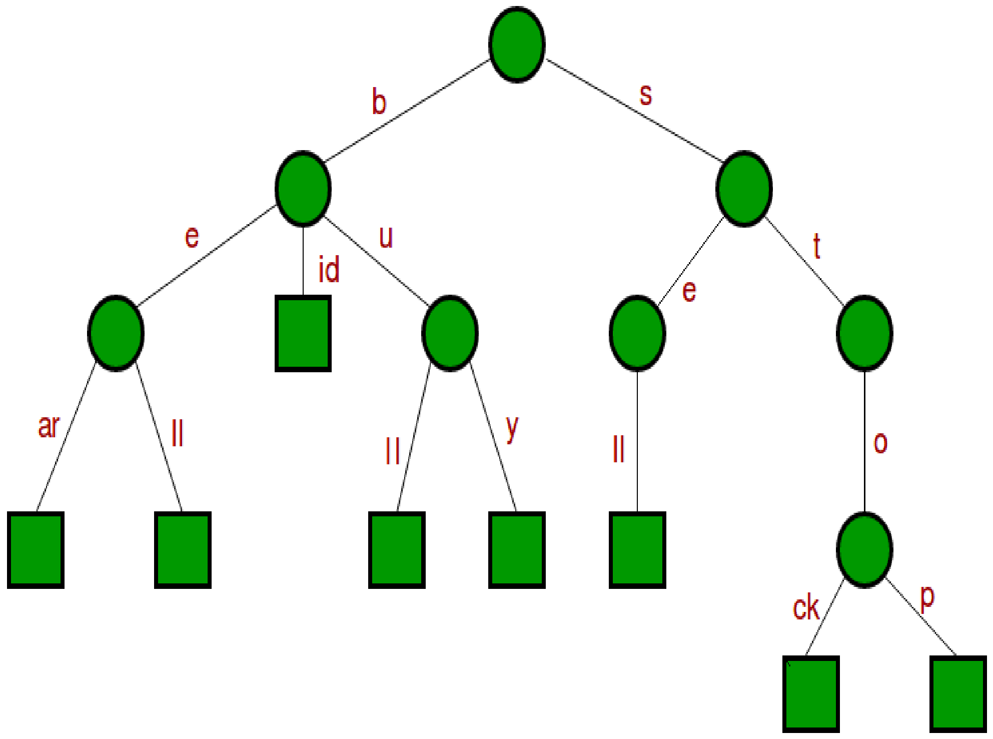
- {bear, bell, bid, bull, buy, sell, stock, stop}
- Following is standard trie for the above input set of words.



# Suffix Tree

- Suffix Tree-Compress Trie is obtained from standard trie by joining chains of single nodes. The nodes of a compressed trie can be stored by storing index ranges at the nodes.





# Josephus Problem

- There are **N** people standing in a circle waiting to be executed. The counting out begins at some point in the circle and proceeds around the circle in a fixed direction.
- In each step, a certain number of people are skipped and the next person is executed.
- The elimination proceeds around the circle (which is becoming smaller and smaller as the executed people are removed), until only the last person remains, who is given freedom.

## Josephus Problem

- Given the total number of persons **N** and a number **k** which indicates that **k-1** persons are skipped and the **kth** person is killed in a circle. The task is to choose the person in the initial circle that survives.

# Josephus Problem

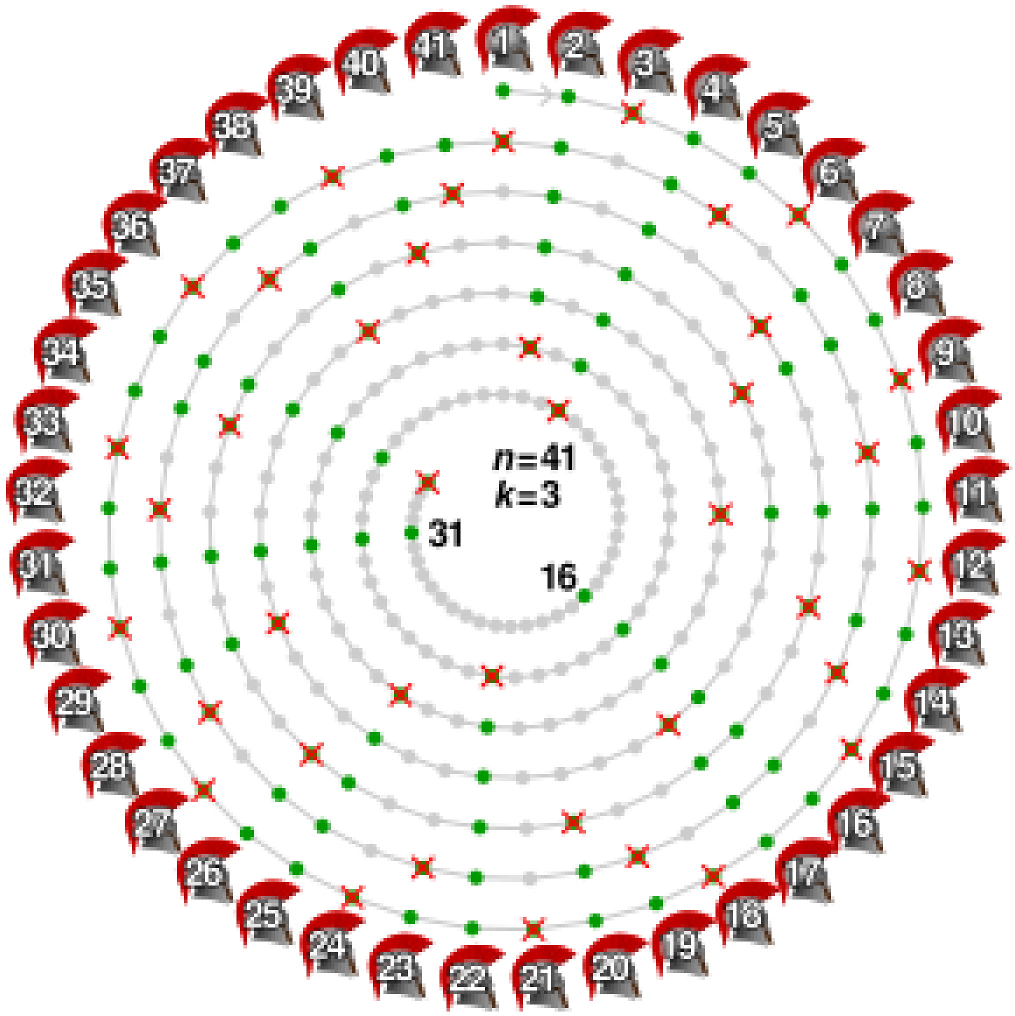


Image Courtesy:https://en.wikipedia.org/wiki/Josephus_problem

## Examples:

- **Input:** N = 5 and k = 2
  **Output:** 3
  **?**

## Examples:

- **Input:** N = 5 and k = 2
  **Output:** 3
  **Explanation:** Firstly, the person at position 2 is killed,
  then the person at position 4 is killed, then the person at position 1 is killed.
  Finally, the person at position 5 is killed. So the person at position 3 survives.

## Examples:

- **Input:** N = 7 and k = 3
  **Output:** 4
  **?**

## Examples:

- **Input:** N = 7 and k = 3
  **Output:** 4
  **Explanations:** The persons at positions 3, 6, 2, 7, 5, and 1 are killed in order,
  and the person at position 4 survives.

# Pseudocode :

```
Josephus( list , start , k){
   if list.size = 1
      return list[0]
   start = (start + k) % list.size
   list.remove( start )
   return Josephus( list, start, k)
}
```

## Josephus Problem

- Josephus Problem can be solved using Circular Queues implemented with Linked list.