Cabinet

by: Mohamed Esmail

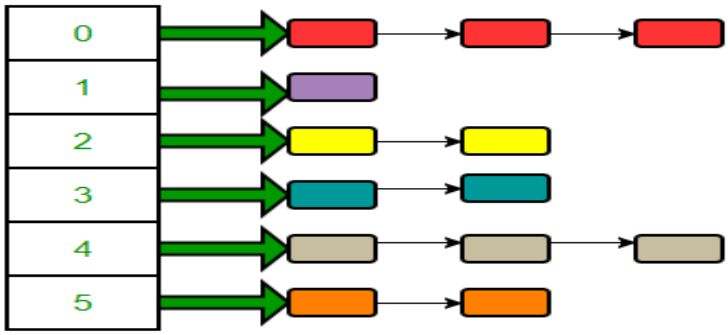
1. **Aim** is to create a structure to speed up the process of inserting and deleting data and make it more organized. For this reason **Cabinet Structure** has built. And this **Cabinet** contain n **Drawers** (n is number of Drawers is Cabinet).

2. Structure:

Cabinet is an **Array**(1d,2d,3d..) of Drawers.

Each Draw is a type of any **List (Linked list, Sorted Linked List, Stack, Queue, tree…)** of Integers, Strings or any data type or Object.

If we have a data set, we can arrange it according to a mathematical expression or according the type of data (String, character, integer, Cat, Dog, Car,…).



- **Example:**

Suppose this data set {A, 3, Text, 1, C, ABC, 2, 4, B}And this is the arrange of this data according to type of data if is a Integer or a Character or a String

Draw[0] Integer	Draw[1] Character	Draw[2] String
1 2 3 4	'A' 'B' 'C'	"ABC" "Text"

- **Another Example:**

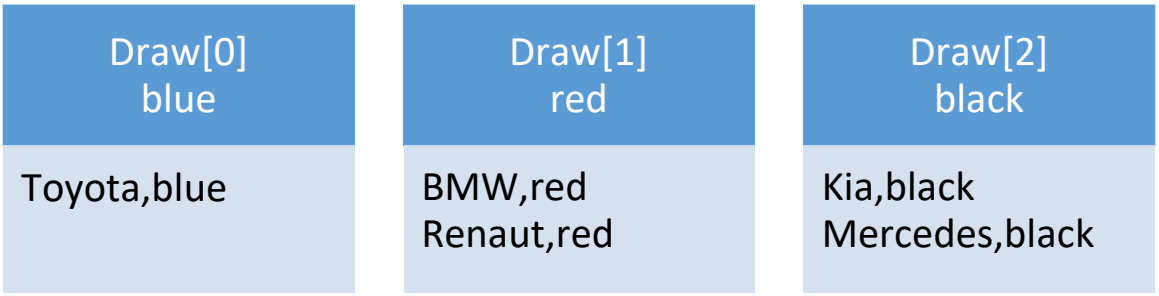
  We can also arrange same type of Object according for some details

  For **example**:

  We can arrange cars according on their colors

  We suppose we have these cars:

  {(BMW,red),(Toyota,blue),(Mercedes,black),(Renaut,red),(Kia,black)}

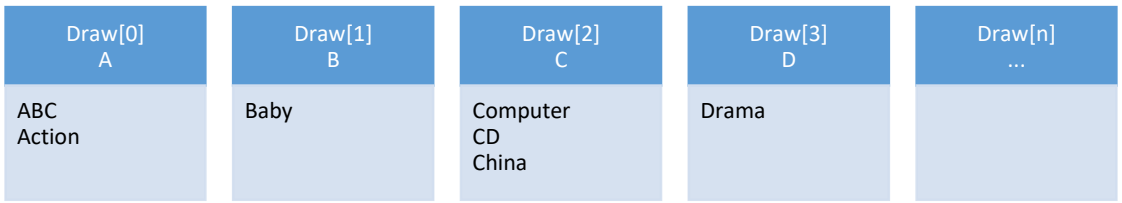


- **Another Example:**

we can store String according of first letter or for first two letter(Dictionary)

we suppose we have this set of data:

{ Abc, Computer, Baby, Action, Drama, Name, Zebra, Cd, China…)



3. Cabinet And Integers:

to **insert, delete, search,…** a number is cabinet use a Mathematic expression to know in which draw we process. We will use **Cabinet** with **Sorted Linked List** .

I. Creating:

To create Cabinet we want to specify number of draw, and if we have not specify it will create one with default number 10

- ***Java implementation:***

```
public Cabinet(int len){
   Drawers=new Node[len];
   this.len=len;
}
```

II. Is Empty:

A draw n is empty mean the draw n equal to null

So to check if a draw if empty we want to check if it equal to null or not

- **Java implementation:**

```
public boolean isEmpty(int draw){
   draw=draw%len;
   if(Drawers[draw]==null)return true;
   return false;
}
```

to check if the Cabinet is empty, we search in all draw and if all are empty so Cabinet is empty, if at least one draw have any data so the Cabinet is not empty

- **java implementation:**

```
public boolean isEmpty(){
   for(int i=0;i<len;i++)
      if(Drawers[i]!=null)return false;
   return true;
}
```

Performance of inserting is O(len) where len is length of cabinet.

III. Inserting:

Number will store at draw = number Mod length of cabinet

For example:

I create a cabinet with 10 draw

And I will insert numbers 4 and 25 and 66

So number 4 will add to the draw number (4%10)=4

And number 25 will add to the draw number (25%10)=5

And number 66 will add to the draw number 6

- **Implementation in java:**

```
public void insert(int data){
int i=data%len;
Node current=Drawers[i];
Node n=new Node(data);
if(current==null||current.getData()>=data){
   n.setNext(current);
   Drawers[i]=n;
}
else{
while(current.getNext()!=null&&current.getNext().getData()<data)
   current=current.getNext();
n.setNext(current.getNext());
current.setNext(n);
}
}
```

Performance of inserting is O(n/len) where n is number of data stored and len is length of cabinet.

IV. Deleting:

a. Delete Data:

If we want to delete a number n from the Cabinet

So firstly we want to find number of draw (n%len) where n is the number and len is the length of the Cabinet. then we work with this draw like a Stored Linked List and we try to find the address of the number. Finally, we delete it.

Example:

We suppose we want to delete an number from a Cabinet with 10 draw for example 7, 13, 56:

so for 7 we search in draw number 7%10=7

and for 13 we search in draw number 13%10=3

and for 56 we search in draw number 56%10=6

and the we try to find the address of the number and we delete it.

- **Java implementation:**

```
public void delete(int data){
   Node current=Drawers[data%len];
   Node previous=null;
   Node first=current;
   while(current!=null){
      if(current.getData()>=data)break;
      else{
      previous=current;
      current=current.getNext();
      }
      }
   if(current.getData()==data){
      if(current==first)Drawers[data%len]=current.getNext();
      else previous.setNext(current.getNext());
   }
}
```

Performance of deleting is O(n/len) where n is number of data stored and len is length of cabinet.

b. Delete from draw from back:

```
public void DeleteFromDrawFromBack(int draw){
   draw=draw%len;
   Node current=Drawers[draw];
   Node previous = null;
   if(current==null||current.getNext()==null)Drawers[draw]=null;
   else{
   while(current.getNext()!=null){
      previous=current;
      current=current.getNext();
   }
   previous.setNext(null);
   }
}
```

c. Delete from draw from front:

```
public void DeleteFromDrawFromFront(int draw){
   draw=draw%len;
   if(Drawers[draw]==null||Drawers[draw].getNext()==null)Drawers[draw]=null;
   else Drawers[draw]=Drawers[draw].getNext();
}
```

d. Delete from draw at position:

```
public void DeleteFromDrawAtPosition(int draw,int p){
   draw=draw%len;
   Node current=Drawers[draw];
   if(p--==0){
      Drawers[draw]=Drawers[draw].getNext();
   }
   else{
   while(p--!=0){
      if(current==null)return;
      current=current.getNext();


   }
   if(current!=null)
   if(current.getNext()!=null)
   current.setNext(current.getNext().getNext());
   else current.setNext(null);
   }
}
```

e. Delete all:

```
public void DeleteAll(){
   for(int i=0;i<len;i++)
      Drawers[i]=null;
}
```

V. Searching:

While that the inserting of a number n, will be insert in the draw n%len.

So to search for n, we will search in the draw number n%len also.

Example:

If we have a cabinet with 10 draw

And we will search for numbers 4 and 25 and 66

So number 4 will be in the draw number (4%10)=4

And number 25 will be in the draw number (25%10)=5

And number 66 will be in the draw number 6

- **Java implementation:**

```
public int Search(int data){
   int count=0;
   Node current=Drawers[data%len];
   while(current!=null){
      if(current.getData()==data)return count;
      current=current.getNext();
      count++;
   }
   return -1;
}
```

Performance of searching is O(n/len) where n is number of data stored and len is length of cabinet.

VI. Display:

We pass to all draw and we print all data that contain.

- **Java implementation:**

```
public void Display(){
   for(int i=0;i<len;i++){
      Node current=Drawers[i];
      System.out.printf("%d :",i);
      while(current!=null){
         System.out.printf("%d  ",current.getData());
         current=current.getNext();
      }
      System.out.println();
   }
}

public void DisplayDraw(int n){
      Node current=Drawers[n];
      while(current!=null){
         System.out.printf("%d  ",current.getData());
         current=current.getNext();
      }
}
```

Performance of inserting is O(n) where n is number of data sorted.

VII. Re-building:

Re-build method is a method to change the change the number of drawers. Like change size of a cabinet from 10 draw to 5 draw.

To rebuild a Cabinet, first we create new cabinet with the new size, and then we put in all data from previous Cabinet.

- **Java implementation:**

```
public void ReBuild(int len){
   Cabinet C=new Cabinet(len);
  for(int i=0;i<this.len;i++){
       Node current=Drawers[i];
      while(current!=null){
         C.insert(current.getData());
         current=current.getNext();
      }
   }
  this.Drawers=C.GetDrawers();
  this.len=len;
}
```

4. Working with Data:

For example prime:

we suppose we want to insert all prime number from 1 till 50

in a Cabinet with 10 draw

Step:

1. Create Cabinet
2. Search all number Prime
3. Add it to the Cabinet
4. Display Cabinet

java implementation:

```
Cabinet C =new Cabinet();
    for(int i=2;i<50;i++)
       if(IsPrime(i))C.inster(i);
       C.Display();
```

algorithm of Prime Number:

```
public static boolean IsPrime(int n){
    for(int i=2;i<=Math.sqrt(n);i++){
       if(n%i==0)return false;
    }
    return true;
  }
```

Performance to check a number is prime or not is O($\sqrt{n-1}$) with this algorithm.

<u>In 10 draw:</u>

draw[0]	draw[1]	draw[2]	draw[3]	draw[4]	draw[5]	draw[6]	draw[7]	draw[8]	draw[9]
	11 31 41 61 71	2	3 13 23 43 53 73 83		5		7 17 37 47 67 97	19 29 59 79 89	

Re-shape first 50 prime number in different size (1-9)

With 1 draw:

Draw[0]
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47

With 2 draw:

draw[0]	draw[1]
2	3 5 7 11 13 17 19 23 29 31 37 41 43 47

With 3 draw:

draw[0]	draw[1]	draw[2]
3	7 13 19 31 37 43	2 5 11 17 23 29 41 47

With 4 draw:

draw[0]	draw[1]	draw[2]	draw[3]
	5 13 17 29 37 41	2	3 7 11 19 23 31 43 47

With 5 draw:

draw[0]	draw[1]	draw[2]	draw[3]	draw[4]
5	11 31 41	2 7 17 37 47	3 13 23 43	19 29

With 6 draw:

draw[0]	draw[1]	draw[2]	draw[3]	draw[4]	draw[5]
	7 13 19 31 37 43	2	3		5 11 17 23 29 41 47

With 7 draw:

draw[0]	draw[1]	draw[2]	draw[3]	draw[4]	draw[5]	draw[6]
7	29 43	2 23 37	3 17 31	11	5 19 47	13 41

With 8 draw:

draw[0	draw[1	draw[2	draw[3	draw[4	draw[5	draw[6	draw[7
	17 41	2	3 11 19 43		5 13 29 37		7 23 31 47

With 9 draw:

draw[0]	draw[1]	draw[2]	draw[3]	draw[4]	draw[5]	draw[6]	draw[7]	draw[8]
	19 37	2 11 29 47	3	13 31	5 23 41		7 43	17

Draw 1

Work like sorted linked list

Draw 2

Odd in draw[1]

Even in draw[0]

Draw 6

all number in 5 and 1 without 2 and 3

New algorithm:

According to Cabinet with 6 draw we can create a new algorithm. Because all number are in draw[1] and draw[5] unless 2 and 3.

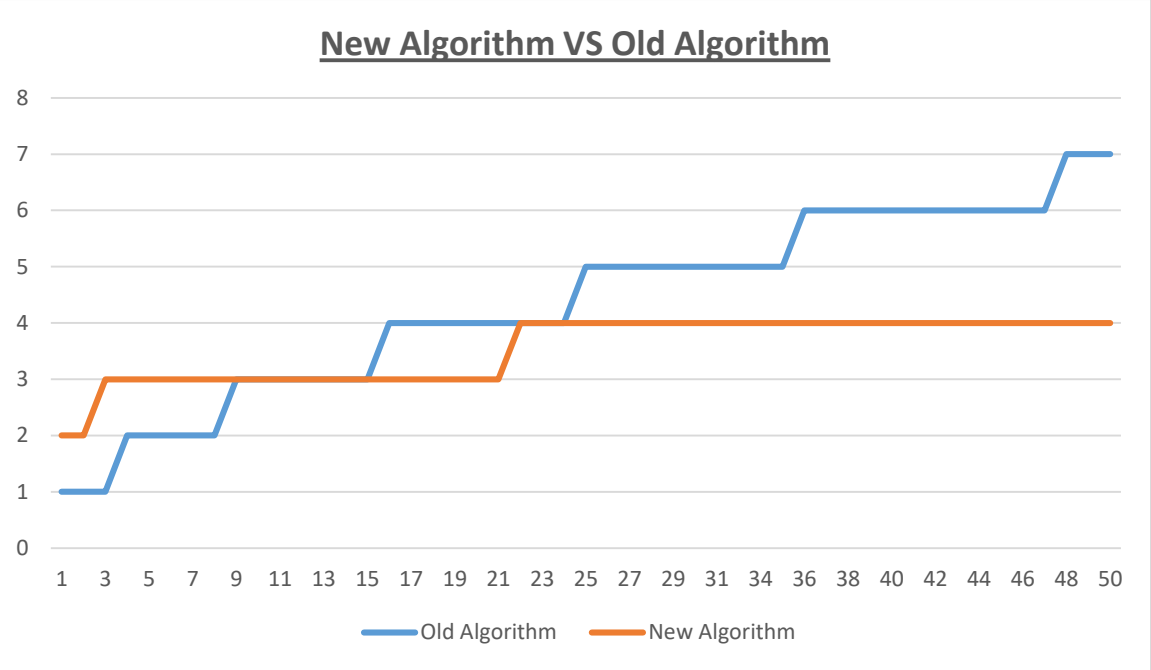
```
public static boolean NewIsPrime(int n){
    if(n==2||n==3)return true;
    else{
       if(n%2==0||n%3==0)return false;
    for(int i=1;i<n-6;){
       i+=4;
       if(n%i==0)return false;
       i+=2;
       if(n%i==0)return false;
    }
    }
    return true;
  }
```

Performance to check a number is prime or not is O($\frac{2\times\sqrt{n}}{6}+2$) with this algorithm.

Old algorithm vs new algorithm:

if we want to check if **100** is prime number using **old algorithm** we want to do $\sqrt{100-1}$**= 9 step.**
but using the **new algorithm** we want to use $\frac{2\times\sqrt{100}}{6}+2$ **= 5 step.**

BUT, if we want to check **2** is prime, we want **1 step** in **old algorithm** and **2 step in new algorithm**.



According to the previous, is better to use **Old algorithm** if number n is less then **9**. And the **New algorithm** is better if n is great then **16**.And we have **same performance** if n is between **[9,16]** and **[22,25]**.

5. **Routine:**

Is a method compute the Entropy of the Cabinet. So it return a number between 0 and 1.

$$\sum_{i=0}^{len} -\frac{count[i]}{C}\times Log(\frac{count[i]}{C})$$

Where count[i] is number of data in each draw, And C is number of data in the Cabinet.

If the Entropy close to 0 so we can say there is a routine is the data. And if is close to 1 so we can say there is a mess in data.

-

- **For example1:**

  We suppose we have a parking and we will insert cars according there colors in a Cabinet:

blue draw[0]	red draw[1]	black draw[2]
Bmw-123 Mercedes-99 Toyota101 Renault-xyz BMW-X		Tesla-i

  Checking if there is a routine:

  - Count[]={5,0,1}
  - C=6

  Entropy= $-\frac{5}{6}\log(\frac{5}{6}) - \frac{0.1}{6}\log(\frac{0.1}{6}) - \frac{1}{6}\log(\frac{1}{6})$= 0.472232

  Entropy = 0.472232 (close to 0). So we can say all car have same color in the parking.

- **Example2:**

blue draw[0]	red draw[1]	black draw[2]
Bmw-123 Mercedes-99 Toyota101	Renault-xyz BMW-X	Tesla-i

  Entropy =0.92061 . close to 1, So there is not routine.

- **java implementation:**

```
public double Routine(){
    double count[]=new double[len];
    double c=0;
    for(int i=0;i<len;i++){
      Node n=Drawers[i];
      while(n!=null){
        n=n.getNext();
        count[i]++;
        c++;
      }
    }
    double entropy=0;
    for(int i=0;i<len;i++){
      if(count[i]!=0) {
        double n = count[i] / c;
        entropy += (-n * (Math.log10(n) / Math.log10(len)));
      }
    }
    return entropy;
  }
```