Bellman Ford Algorithm

This Algorithm is type from Dynamic programming because to try to found short path from source /all Node in graph and if graph contain negative weight that meaning some shortest path maybe not exist because FB is procedure used to find all shortest path in graph to order compare from source with all other nodes in graph so the algorithm not contain any negative cycle length and FB algorithm is rely on the condition is called relax operation and relax procedure takes two nodes as Parameter

Dijkstra not work with negative cycle length but FB is simpler than Dijkstra so time complexity of FB type from DP (O(VE)) more than Dijkstra type from Greedy algorithm (O(VLogV))

IF Distance from the source to First Node (A) + edge.length is less than Distance to the second node than the first node is denoted as the predecessor of the second node and the distance to the second node is recalculated (distance(A) + edge.length)

Detailed steps:

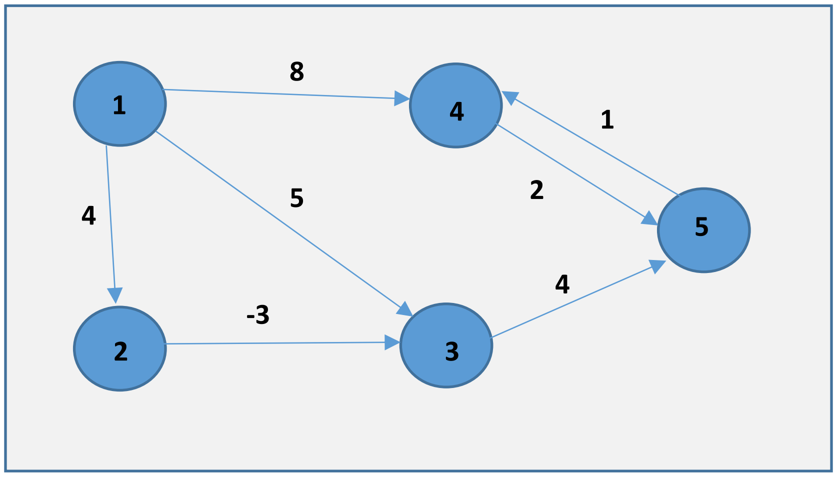
1- This step initializes distances from source to all vertices as infinite
2- Distance to source itself equal 0 and create array dist [ ] of size vertex with all values is infinite except the distance to source itself and must total number of vertices in the graph is for example 5 so all edges must be processed 4 times
3- This step calculates shortest distances
If distances [vertex] > distances [source] + weight of edge then update distances [source]
distances [vertex] = distances [source] + weight of edge

4- This step reports if there is NCL in graph.
If distances [vertex] > distances [source] + weight of edge
Then error its graph contains negative weight cycle

The detailed of step 4 is step 3 if graph doesn't contain NCL

Example:

The graph may contain negative weight edges and Dijkstra algorithm is greedy algorithm and time complexity is O(vlogv) so Dijkstra doesn't work for graph s with negative weight edges and Bellman works for each for such graph and it's also simpler than Dijkstra and suit well for distribution systems but time complexity of ford bellman is O(VE) which is more than Dijkstra



Random Order	
U	V
4 $\xrightarrow{2}$	5
5 $\xrightarrow{1}$	4
3 $\xrightarrow{4}$	5
1 $\xrightarrow{5}$	3
2 $\xrightarrow{-3}$	3
1 $\xrightarrow{8}$	4
1 $\xrightarrow{4}$	2

Vertex	Distance	Parent
1	0	-
2	∞	-
3	∞	-
4	∞	-
5	∞	-

Complexity:

Space: O (V)
Time: O (V.E)
Short Path: O(V2.E)
Short Path in worst Case: O(V4.E)

Start v = 1:

4 $\longrightarrow$ 5 = INF + 2 < ∞ - wrong X

1 $\longrightarrow$ 3 = 0 + 5 < 2 - correct √ -> Then vertex = 3 dis = 5 parent = 1

5 $\longrightarrow$ 4 = INF + 2 < ∞ - wrong X

1 $\longrightarrow$ 4 = 0 + 8 < ∞ - correct √ -> Then vertex = 4 dis = 8 parent = 1

5 $\longrightarrow$ 4 = 10 + 1 < 8 - wrong X

Start v = 2:

4 $\longrightarrow$ 5 = 8 + 2 < ∞ - correct √ -> Then vertex = 5 dis = 10 parent = 4

1 $\longrightarrow$ 3 = 0 + 5 < 5 - wrong X

3 $\longrightarrow$ 5 = 5 + 4 < 10 - correct √ -> Then vertex = 5 dis = 9 parent = 3

1 $\longrightarrow$ 4 = 0 + 8 < 8 - wrong X

2 $\longrightarrow$ 3 = 4 - 3 < 5 - correct √ -> Then vertex = 3 dis = 1 parent = 2

1 $\longrightarrow$ 2 = 0 + 4 < 4 - wrong X

5 $\longrightarrow$ 4 = 9 +1 < 8 - wrong X

Start v = 3:

4 $\longrightarrow$ 5 = 8 + 2 < 9 - wrong X

3 $\longrightarrow$ 5 = 1 + 4 < 9 - correct √ -> Then vertex = 5 dis = 5 parent = 3

1 $\longrightarrow$ 3 = 0 + 5 < 1 - wrong X

2 $\longrightarrow$ 3 = 4 - 3 < 1 - wrong X

1 $\longrightarrow$ 4 = 0 + 8 < 8 - wrong X

1 $\longrightarrow$ 5 = 0 + 1 < 5 - correct √ -> Then vertex = 5 dis = 5 parent = 1

Start v = 4:

4 $\longrightarrow$ 5 = 8 + 2 < 9 - wrong X

5 $\longrightarrow$ 4 = 5 + 1 < 8 - correct √ -> Then vertex = 4 dis = 6 parent = 5

3 $\longrightarrow$ 5 = 1 + 4 < 5 - wrong X

1 $\longrightarrow$ 3 = 0 + 5 < 1 - wrong X

2 $\longrightarrow$ 3 = 4 - 3 < 1 - wrong X

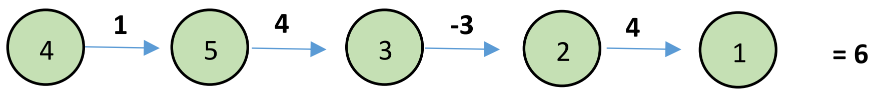
1 $\longrightarrow$ 4 = 0 + 8 < 6 - wrong X

1 $\longrightarrow$ 2 = 0 + 4 < 4 - wrong X

After Finished All Iteration (v)

Vertex	Distance	Parent
1	0	Null
2	4	0
3	1	1
4	6	4
5	5	2

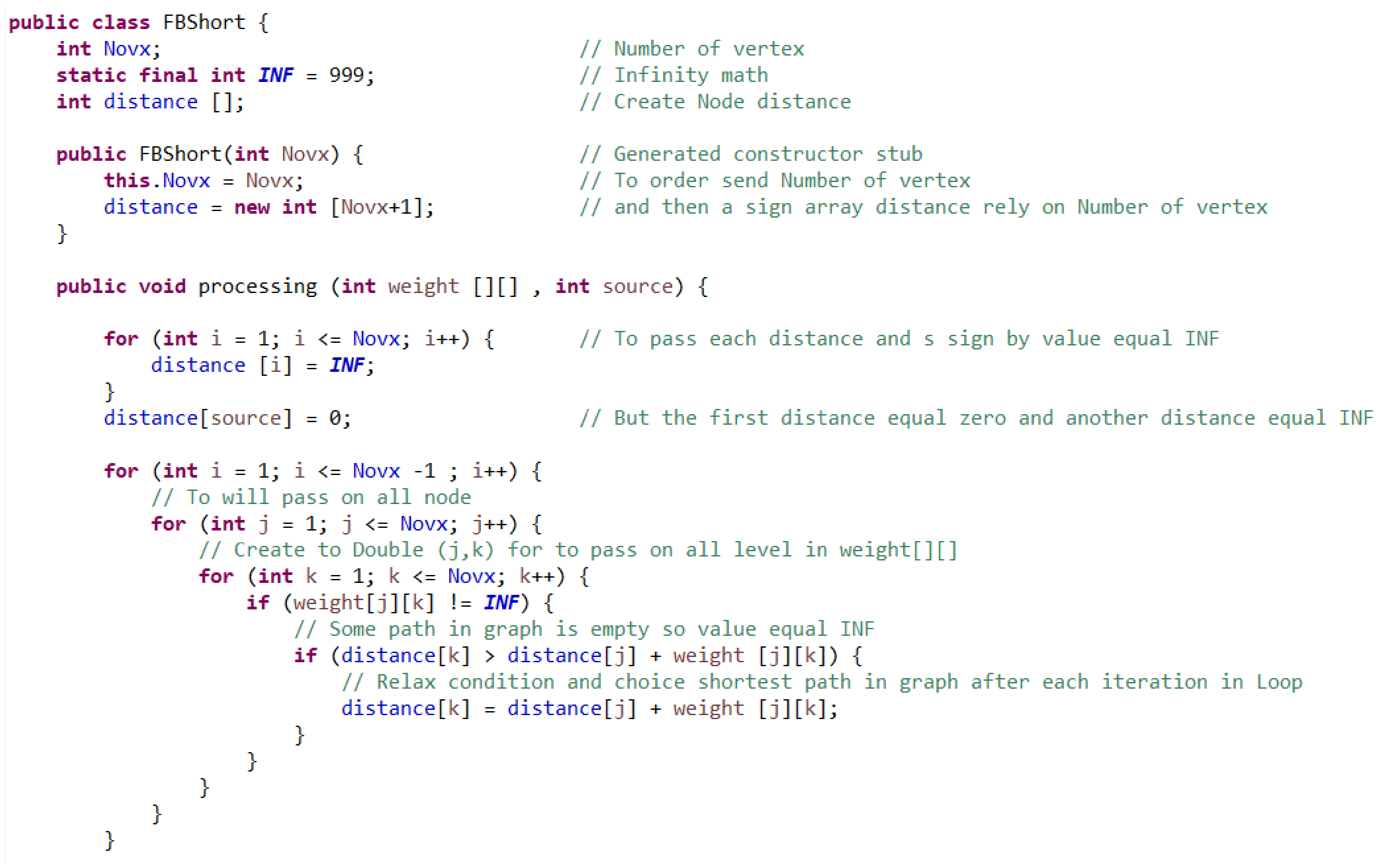
For example if you need Go to 4



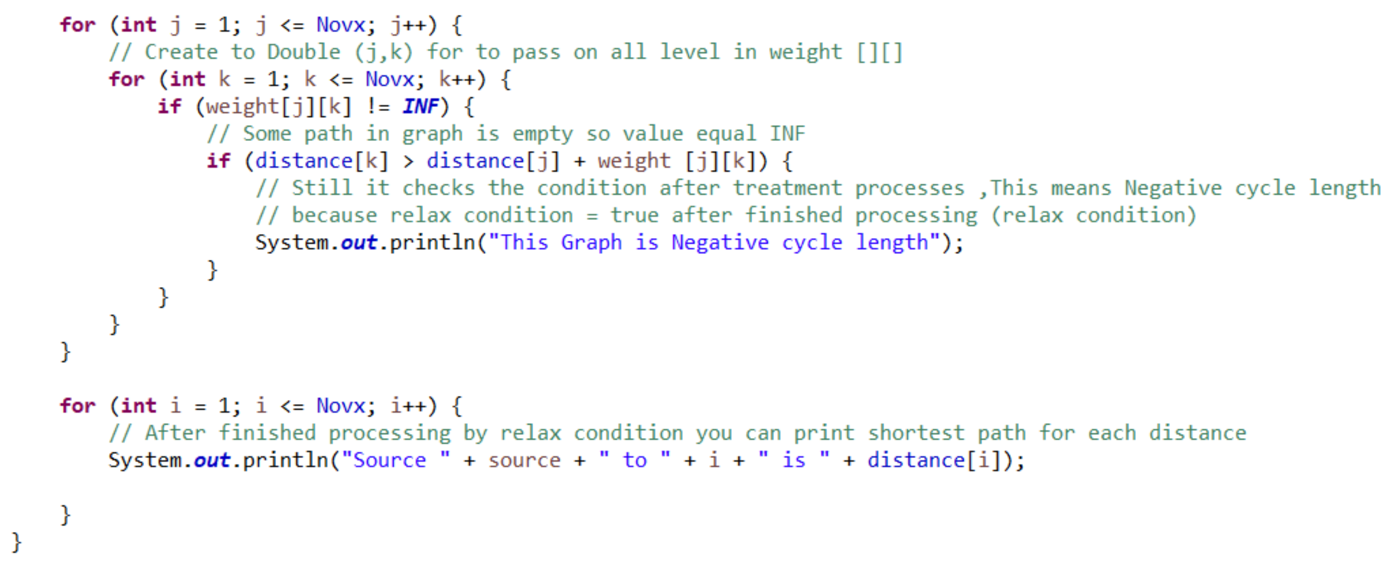
Bellman Ford Pseudo Code

```
Public void FB (int start, int weight [ ][ ]) {

        For (int i = 1; i <= Nonx; i++){
              distance [i] = INF
        }
        distance [start] = 0;

        For (int i = 1; i <= Novx - 1; i++) {

            For (int j = 1; j <= Novx; j++) {

               For (int k = 1; k <= Novx; k ++) {

                   If (next [j][k] != INF) {

                     If (distance [k] > distances [j] + next [j][k])

                       distance [k] = distances [j] + next [j][k];  }

                   }
                }
            }

          For (int j = 1; j <= Novx; j++) {

                 For (int k = 1; k <= Novx; k++){

                   If (next [j][k] != INF) {

                    If (distance [k] > distances [j] +next [j][k]

                         system.out.println("This Graph is NCL");

                    }
                 }
              }
```

Implementation Code



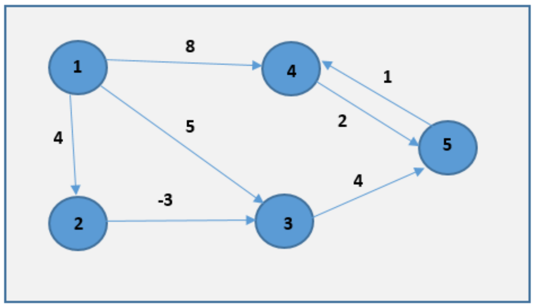
```java
public class FBShort {
    int Novx;                                   // Number of vertex
    static final int INF = 999;                 // Infinity math
    int distance [];                            // Create Node distance

    public FBShort(int Novx) {                  // Generated constructor stub
        this.Novx = Novx;                       // To order send Number of vertex
        distance = new int [Novx+1];            // and then a sign array distance rely on Number of vertex
    }

    public void processing (int weight [][] , int source) {

        for (int i = 1; i <= Novx; i++) {       // To pass each distance and s sign by value equal INF
            distance [i] = INF;
        }
        distance[source] = 0;                   // But the first distance equal zero and another distance equal INF

        for (int i = 1; i <= Novx -1 ; i++) {
            // To will pass on all node
            for (int j = 1; j <= Novx; j++) {
                // Create to Double (j,k) for to pass on all level in weight[][]
                for (int k = 1; k <= Novx; k++) {
                    if (weight[j][k] != INF) {
                        // Some path in graph is empty so value equal INF
                        if (distance[k] > distance[j] + weight [j][k]) {
                            // Relax condition and choice shortest path in graph after each iteration in Loop
                            distance[k] = distance[j] + weight [j][k];
                        }
                    }
                }
            }
        }
```

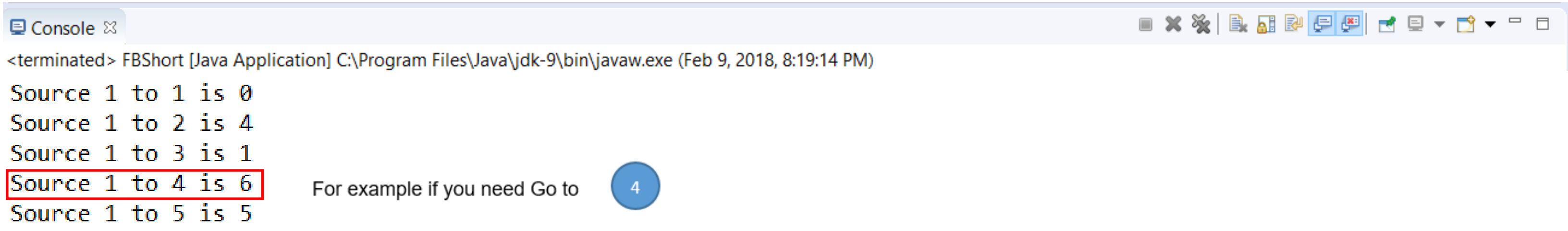


```java
    for (int j = 1; j <= Novx; j++) {
        // Create to Double (j,k) for to pass on all level in weight [][]
        for (int k = 1; k <= Novx; k++) {
            if (weight[j][k] != INF) {
                // Some path in graph is empty so value equal INF
                if (distance[k] > distance[j] + weight [j][k]) {
                    // Still it checks the condition after treatment processes ,This means Negative cycle length
                    // because relax condition = true after finished processing (relax condition)
                    System.out.println("This Graph is Negative cycle length");
                }
            }
        }
    }

    for (int i = 1; i <= Novx; i++) {
        // After finished processing by relax condition you can print shortest path for each distance
        System.out.println("Source " + source + " to " + i + " is " + distance[i]);

    }
}
```

Implementation Code

```java
public static void main(String[] args) {

        int Novx = 5;
        int source = 1;
        int weight[][]= {{0,4,5,8,0},
                         {0,0,-3,0,0},
                         {0,0,0,0,4},
                         {0,0,0,0,2},
                         {0,0,0,1,0}};

        for (int j= 1; j <= Novx; j++) {
            for (int k = 1; k <= Novx; k++) {
                if (j == k) {
                    // Not available to cross array in FB algorithms and it's meaning not available
                    // Turn around itself node in Graph so all number in cross array must equal zero
                    // and when number column equal number row in cross array
                    weight[j][k] = 0 ;
                    continue;
                }
                if (weight[j][k] == 0) {
                    // some distance in graph empty and value equal zero so you must change zero to INF
                    // In order not to be done treatment process by relax condition
                    weight[j][k] = INF ;
                }
            }
        }

        FBShort o = new FBShort(Novx);
        o.processing(weight, source);
    }
}
```



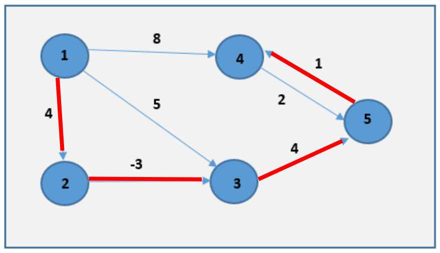

Console

```
<terminated> FBShort [Java Application] C:\Program Files\Java\jdk-9\bin\javaw.exe (Feb 9, 2018, 8:19:14 PM)
Source 1 to 1 is 0
Source 1 to 2 is 4
Source 1 to 3 is 1
Source 1 to 4 is 6
Source 1 to 5 is 5
```

For example if you need Go to 4

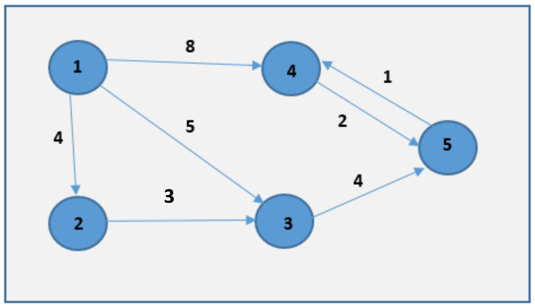
$$\begin{tikzcd} 4 \arrow[r, "1"] & 5 \arrow[r, "4"] & 3 \arrow[r, "-3"] & 2 \arrow[r, "4"] & 1 & \boxed{=6} \end{tikzcd}$$

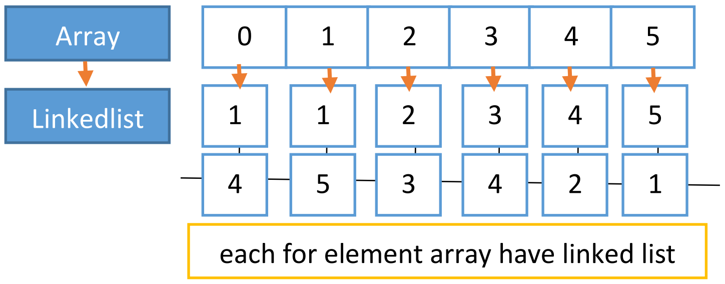
Vertex	Distance	Parent
1	0	Null
2	4	0
3	1	1
4	6	4
5	5	2



```
Class Node      {
public:
            int data;
            Node *next;
            Node *prev;
}*head, cursor;

Bool curIsEmpty() const {
         return (cursor == NULL);
}

Bool atEnd() const {
if ( listIsEmpty()) return true;
      else if (curIsEmpty()) return false;
                 else return (cursor->next == NULL) && (head->prev == NULL) ;
}

void Createnew()        {
            Node *newptr;
            int data[2] = {number , weight} ;
            newptr = new Node;
            cout << "Enter number: ";
            cin >> data[0];
            cout << "Enter weight: ";
            cin >> data[1];
            newptr -> data=data;
            newptr -> next=NULL;
            newptr -> prev=NULL;
            return newptr , data;
}

void insertfirst()          {
            Node *newnode;
            newnode = Createnew();

            if(head == NULL)      {
                        newnode -> next = newnode;
                        newnode -> prev = newnode;
                        head = newnode;
            }

            else      {
            Node *temp;
            temp = head;
            newnode -> next = temp;
            newnode -> prev = temp -> prev;
            temp -> prev -> next = newnode;
            temp -> prev = newnode;
            head = newnode;
            }
}

void insertlast()           {
            Node *newnode;
            newnode = createnew();
            if(head == NULL)      {
                        head = newnode;
                        newnode -> next = newnode;
                        newnode -> prev = newnode;
            }
            else      {
                        Node *temp;
                        temp = head;
                        temp = temp -> prev;
                        newnode -> next = head;
                        newnode -> prev = temp;
                        temp ->next = newnode;
                        head -> prev = n ewnode;
            }
}
void insertcenter() {
            int no;
            if (no == 0)      {
               insertfirst();
            return;
            }
            Node *newnode,*p,*l;
            newnode = Createnew();
            p = head;
            for (int i=1; i<no ; i++)
                  p = p-> next;
            l = p -> next;
            p -> next = newnode;
            newnode -> next = l;
            l -> prev = newnode;
            newnode -> prev = p;
}
```

```
void swapdata()         {
        if(head == NULL) {
                return;
        }
        Node *temp,*p;
        temp = head;
        p=head -> prev;
        while(temp!=p)          {
                swap(temp -> data,p -> data);
                temp = temp -> next;
                if(temp == p) return;
                p = p -> prev;
        }
}
int main()      {
int data[] = {node0 , node1 , node2 , node3 , node4};
int node0[] = Createnew();
if (curIsEmpty ()) {
     insertfirst();
} else {
int node1[] = Createnew();
insertcenter();
int node2[] = Createnew();
insertcenter();
int node3[] = Createnew();
insertcenter();
int node4[] = Createnew();
   if (atEnd()) {
      insertlast();}
}
}
for (int i = 0 ; i < 2 ; i++)
    for (int node = 0 ; node < data.lenght < node++){
       if (data[node] -> next == data[node+1] -> prev)
          if (node[0] > node[0] + node[1]) {
             node[0] == node[0] + node[1]

  } else {
     curser = node->next
          if ( !curIsEmpty ())
              swapdata();
          }
else {
      count << “finished processing’’<<
      }
}  }
```


1
2
3
4
5
8
1
4
5
2
3
4


Array
Linkedlist
0
1
2
3
4
5
1
1
2
3
4
5
4
5
3
4
2
1
each for element array have linked list