**Module Name :** Data structure & Algorithm with Java **Date :**

**Max Marks :** 100 Marks

**Duration :** 2 Hours



**Question 1:**

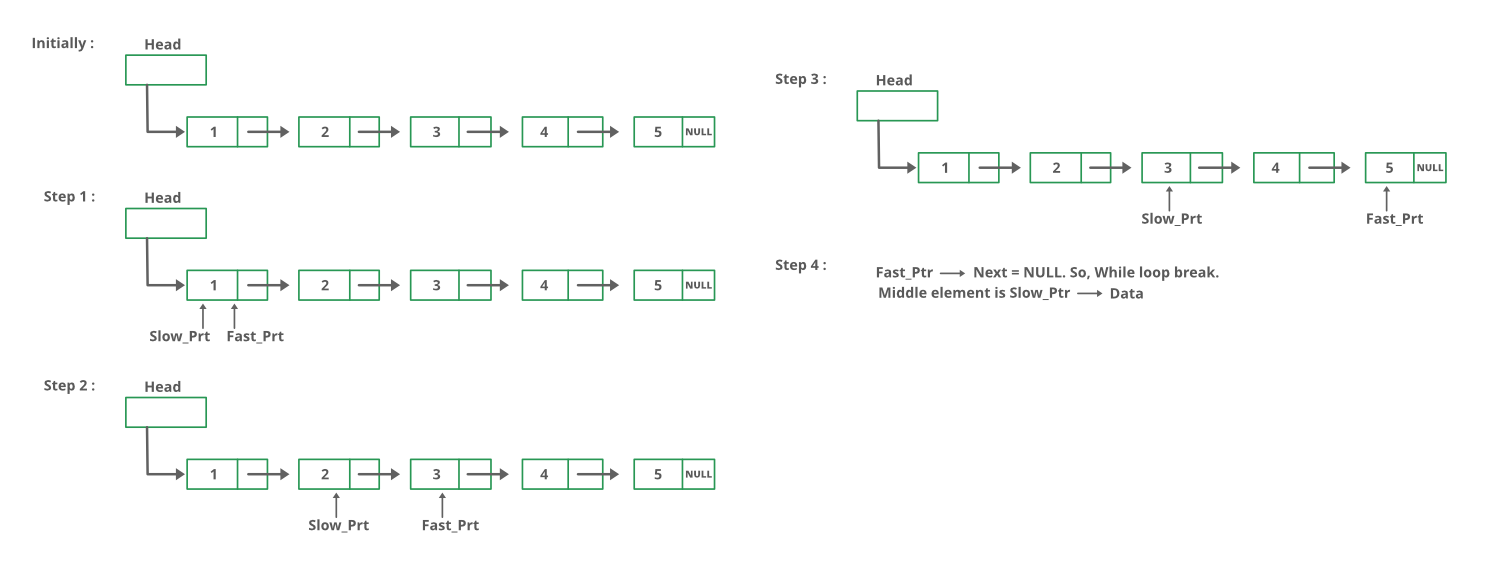
Given a singly linked list, find the middle of the linked list. For example, if the given linked list is 1->2->3->4->5 then the output should be 3.

If there are even nodes, then there would be two middle nodes, we need to print the second middle element. For example, if given linked list is 1->2->3->4->5->6 then the output should be 4.

**Solution**

Traverse linked list using two pointers. Move one pointer by one and the other pointers by two. When the fast pointer reaches the end slow pointer will reach the middle of the linked list.

Below image shows how printMiddle function works in the code :



| // Java program to find middle of linked list class LinkedList {  Node head; // head of linked list   /\* Linked list node \*/  class Node  {  int data;  Node next;  Node(int d)  {  data = d;  next = null;  }  }   /\* Function to print middle of linked list \*/  void printMiddle()  {  Node slow\_ptr = head;  Node fast\_ptr = head;    while (fast\_ptr != null && fast\_ptr.next != null)  {  fast\_ptr = fast\_ptr.next.next;  slow\_ptr = slow\_ptr.next;  }  System.out.println("The middle element is [" +  slow\_ptr.data + "] \n");    }   /\* Inserts a new Node at front of the list. \*/  public void push(int new\_data)  {  /\* 1 & 2: Allocate the Node &  Put in the data\*/  Node new\_node = new Node(new\_data);   /\* 3. Make next of new Node as head \*/  new\_node.next = head;   /\* 4. Move the head to point to new Node \*/  head = new\_node;  }   /\* This function prints contents of linked list  starting from the given node \*/  public void printList()  {  Node tnode = head;  while (tnode != null)  {  System.out.print(tnode.data+"->");  tnode = tnode.next;  }  System.out.println("NULL");  }   public static void main(String [] args)  {  LinkedList llist = new LinkedList();  for (int i=5; i>0; --i)  {  llist.push(i);  llist.printList();  llist.printMiddle();  }  } } |
| --- |



**Question 2:**

Convert Infix to prefix

Example:

**Input**: A \* B + C / D

**Output**: + \* A B/ C D

**Input**: (A - B/C) \* (A/K-L)

**Output**: \*-A/BC-/AKL

**Solution:**

| // JAVA program to convert infix to prefix import java.util.\*; class GFG { static boolean isalpha(char c) {  if (c >= 'a' && c <= 'z' || c >= 'A' && c <= 'Z') {  return true;   }  return false; }  static boolean isdigit(char c) {  if (c >= '0' && c <= '9'){  return true;  }  return false; }  static boolean isOperator(char c) {  return (!isalpha(c) && !isdigit(c)); }  static int getPriority(char C) {  if (C == '-' || C == '+')  return 1;  else if (C == '\*' || C == '/')  return 2;  else if (C == '^')  return 3;  return 0; }  // Reverse the letters of the word static String reverse(char str[], int start, int end) {   // Temporary variable to store character  char temp;  while (start < end) {  // Swapping the first and last character  temp = str[start];  str[start] = str[end];  str[end] = temp;  start++;  end--;  }  return String.valueOf(str); }  static String infixToPostfix(char[] infix1) {  System.out.println(infix1);  String infix = '(' + String.valueOf(infix1) + ')';   int l = infix.length();  Stack<Character> char\_stack = new Stack<>();  String output="";   for (int i = 0; i < l; i++) {   // If the scanned character is an  // operand, add it to output.  if (isalpha(infix.charAt(i)) || isdigit(infix.charAt(i)))  output += infix.charAt(i);  // If the scanned character is an  // '(', push it to the stack.  else if (infix.charAt(i) == '(')  char\_stack.add('(');  // If the scanned character is an  // ')', pop and output from the stack  // until an '(' is encountered.  else if (infix.charAt(i) == ')') {  while (char\_stack.peek() != '(') {  output += char\_stack.peek();  char\_stack.pop();  }  // Remove '(' from the stack  char\_stack.pop();  }   // Operator found  else {  if (isOperator(char\_stack.peek())) {  while ((getPriority(infix.charAt(i)) <  getPriority(char\_stack.peek()))  || (getPriority(infix.charAt(i)) <=  getPriority(char\_stack.peek())  && infix.charAt(i) == '^'))  {  output += char\_stack.peek();  char\_stack.pop();  }  // Push current Operator on stack  char\_stack.add(infix.charAt(i));  }  }  }  while(!char\_stack.empty()){  output += char\_stack.pop();  }  return output; }  static String infixToPrefix(char[] infix) {  /\*  \* Reverse String Replace ( with ) and vice versa Get Postfix Reverse Postfix \*  \*/  int l = infix.length;   // Reverse infix  String infix1 = reverse(infix, 0, l - 1);  infix = infix1.toCharArray();   // Replace ( with ) and vice versa  for (int i = 0; i < l; i++)  {   if (infix[i] == '(')  {  infix[i] = ')';  i++;  }  else if (infix[i] == ')')  {  infix[i] = '(';  i++;  }  }   String prefix = infixToPostfix(infix);   // Reverse postfix  prefix = reverse(prefix.toCharArray(), 0, l-1);   return prefix; }  // Driver code public static void main(String[] args) {  String s = ("x+y\*z/w+u");  System.out.print(infixToPrefix(s.toCharArray())); } } |
| --- |



**Question 3:**

Given a Queue consisting of first **n** natural numbers (in random order). The task is to check whether the given Queue elements can be arranged in increasing order in another Queue using a stack. The operation allowed are:

1. Push and pop elements from the stack

2. Pop (Or Dequeue) from the given Queue.

3. Push (Or Enqueue) in another Queue.

Solution

| // Java Program to check if a queue // of first n natural number can // be sorted using a stack import java.io.\*; import java.util.\*;  class GFG {  static Queue<Integer> q =  new LinkedList<Integer>();    // Function to check if given  // queue element can be sorted  // into another queue using a stack.  static boolean checkSorted(int n)  {  Stack<Integer> st =  new Stack<Integer>();  int expected = 1;  int fnt;    // while given Queue  // is not empty.  while (q.size() != 0)  {  fnt = q.peek();  q.poll();    // if front element is  // the expected element  if (fnt == expected)  expected++;    else  {  // if stack is empty,  // push the element  if (st.size() == 0)  {  st.push(fnt);  }    // if top element is less than  // element which need to be  // pushed, then return false.  else if (st.size() != 0 &&  st.peek() < fnt)  {  return false;  }    // else push into the stack.  else  st.push(fnt);  }    // while expected element are  // coming from stack, pop them out.  while (st.size() != 0 &&  st.peek() == expected)  {  st.pop();  expected++;  }  }    // if the final expected element  // value is equal to initial Queue  // size and the stack is empty.  if (expected - 1 == n &&  st.size() == 0)  return true;    return false;  }    // Driver Code  public static void main(String args[])  {  q.add(5);  q.add(1);  q.add(2);  q.add(3);  q.add(4);    int n = q.size();   if (checkSorted(n))  System.out.print("Yes");  else  System.out.print("No");  } } |
| --- |



**Question 4:**

Given an integer array and a positive integer k, count all distinct pairs with differences equal to k.

**Examples:**

**Input**: arr[] = {1, 5, 3, 4, 2}, k = 3

**Output**: 2

There are 2 pairs with difference 3, the pairs are {1, 4} and {5, 2}

**Input**: arr[] = {8, 12, 16, 4, 0, 20}, k = 4

**Output**: 5

There are 5 pairs with difference 4, the pairs are {0, 4}, {4, 8},

{8, 12}, {12, 16} and {16, 20}

Sol

Algorithm:

Sort the array arr

Take two pointers, l, and r, both pointing to 1st element

Take the difference arr[r] – arr[l]

* If value diff is K, increment count and move both pointers to next element
* if value diff > k, move l to next element
* if value diff < k, move r to next element

return count

| // A sorting based Java program to  // count pairs with difference k import java.util.\*;  class GFG {  /\* Returns count of pairs with difference k in arr[] of size n. \*/ static int countPairsWithDiffK(int arr[], int n,  int k) {  int count = 0;  Arrays.sort(arr); // Sort array elements   int l = 0;  int r = 0;  while(r < n)  {  if(arr[r] - arr[l] == k)  {  count++;  l++;  r++;  }  else if(arr[r] - arr[l] > k)  l++;  else // arr[r] - arr[l] < sum  r++;  }   return count; }  // Driver program to test above function public static void main(String[] args) {  int arr[] = {1, 5, 3, 4, 2};  int n = arr.length;  int k = 3;  System.out.println("Count of pairs with given diff is " +  countPairsWithDiffK(arr, n, k)); } } |
| --- |



**Question 5:**

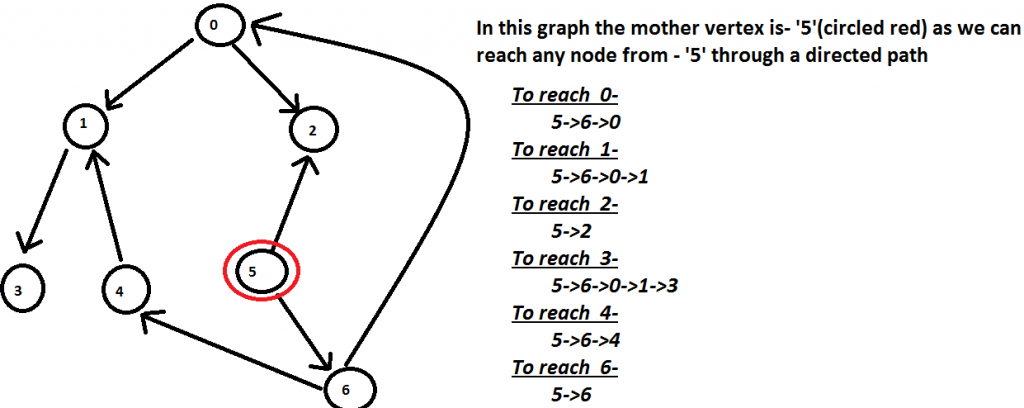
What is a Mother Vertex?

A mother vertex in a graph G = (V, E) is a vertex v such that all other vertices in G can be reached by a path from v.

**Example**:

**Input**: Below Graph

**Output**: 5



**Algorithm :**

1. Do DFS traversal of the given graph. While doing traversal keep track of last finished vertex ‘v’. This step takes O(V+E) time.
2. If there exist mother vertex (or vertices), then v must be one (or one of them). Check if v is a mother vertex by doing DFS/BFS from v. This step also takes O(V+E) time.

| Solution // Java program to find a mother // vertex in O(V+E) time import java.util.\*;  class GFG{   static void addEdge(int u, int v, ArrayList<ArrayList<Integer>> adj) {  adj.get(u).add(v); }  // A recursive function to print DFS starting from v static void DFSUtil(ArrayList<ArrayList<Integer>> g,  int v, boolean[] visited) {  // Mark the current node as  // visited and print it  visited[v] = true;    // Recur for all the vertices  // adjacent to this vertex  for(int x : g.get(v))  {  if (!visited[x])  {  DFSUtil(g, x, visited);  }  } }  // Returns a mother vertex if exists. // Otherwise returns -1 static int motherVertex(ArrayList<ArrayList<Integer>>g,  int V) {    // visited[] is used for DFS. Initially  // all are initialized as not visited  boolean[] visited = new boolean[V];    // To store last finished vertex  // (or mother vertex)  int v = -1;    for(int i = 0; i < V; i++)  {  if (!visited[i])  {  DFSUtil(g, i, visited);  v = i;  }  }    // If there exist mother vertex (or vertices)  // in given graph, then v must be one  // (or one of them)   // Now check if v is actually a mother  // vertex (or graph has a mother vertex).  // We basically check if every vertex  // is reachable from v or not.   // Reset all values in visited[] as false  // and do DFS beginning from v to check  // if all vertices are reachable from  // it or not.  boolean[] check = new boolean[V];  DFSUtil(g, v, check);  for(boolean val : check)  {  if (!val)  {  return -1;  }  }  return v; }  // Driver code public static void main(String[] args) {  int V = 7;  int E = 8;    ArrayList<  ArrayList<Integer>> adj = new ArrayList<  ArrayList<Integer>>();  for(int i = 0; i < V; i++)  {  adj.add(new ArrayList<Integer>());  }  addEdge(0, 1,adj);  addEdge(0, 2,adj);  addEdge(1, 3,adj);  addEdge(4, 1,adj);  addEdge(6, 4,adj);  addEdge(5, 6,adj);  addEdge(5, 2,adj);  addEdge(6, 0,adj);    System.out.println("The mother vertex is " +  motherVertex(adj, V)); } } |
| --- |

**Evaluation of Lab Exam should be based on the following criteria:**

| **Criteria** | **Details** | **Max Marks** | **Marks Obtain** |
| --- | --- | --- | --- |
|  | Documentation of Algorithm and Flowchart |  |  |
| Program adheres to the algorithm and flowchart |
| Efficiency | Program is using only the required number of variables /conditions/loops/pointers etc and is optimal |
|  | The program produces desired output for a given input |
| The program handles all valid and Invalid inputs |
|  | The program has meaning variable/function names |
| The program is commented properly (At least 20% of the code should be commented) |
| total |  | 100 |  |
|  | ***Total Marks*** | **100** |  |

**Signature of Student**