CENG2001- 2020 – Final - Report

23-Ocak-2021

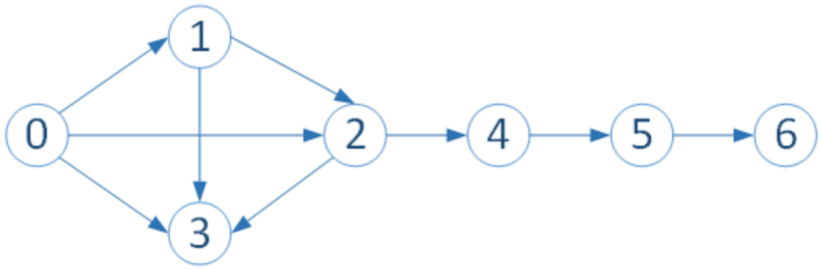
Name : Hasan Ali ÖZKAN
ID : 180709020

Challenge: Directed Acyclic Graphs

Time Complexity: O(V+E)

Example Input:

```
0 1
0 2
0 3
1 2
1 3
2 3
2 4
4 5
5 6
```



Solution Code Trace:

```
58   public static boolean isDag(DirectedGraph digraph) {
59        Map<Integer,Integer> inDeg= digraph.inDegree();
60        Stack<Integer> graphStack = new Stack<>();
61        for (Integer i : inDeg.keySet()){
62            if (inDeg.get(i) == 0) graphStack.push(i);
63        }
64        List<Integer> topSort = new ArrayList<>();
65        while (!graphStack.isEmpty()){
66            Integer temp = graphStack.pop();
67            topSort.add(temp);
68            for (Integer neighbour : digraph.adjList.get(temp)){
69                inDeg.put(neighbour,inDeg.get(neighbour)-1);
70                if (inDeg.get(neighbour)==0) graphStack.push(neighbour);
71            }
72        }
73        if(topSort.size() != digraph.adjList.size()) return false;
74        return true;
75    }
```

adjList ==> {0:[1,2,3], 1:[2,3], 2:[3,4], 3:[], 4:[5], 5:[6], 6:[]}

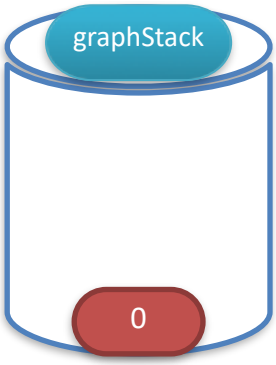
inDeg ==> {0:0, 1:1, 2:2, 3:3, 4:1, 5:1, 6:1}

i = 0{

inDeg.get(0) == 0 then

Push the current key into the graphStack.

}



graphStack is not empty {

temp = 0

Then add to tem into the topSort.

topSort = {0}

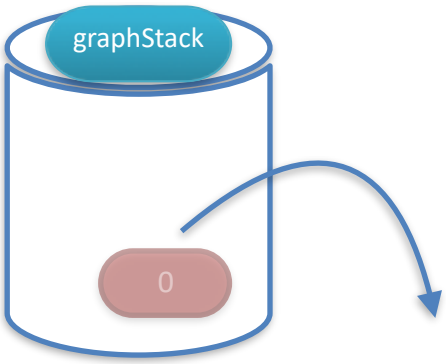
neighbour = 0{

Decrement by 1 the neighbours value in the inDeg.

inDeg ==> {0:-1, 1:1, 2:2, 3:3, 4:1, 5:1, 6:1} then,

-1 = 0 false then do nothing.

}



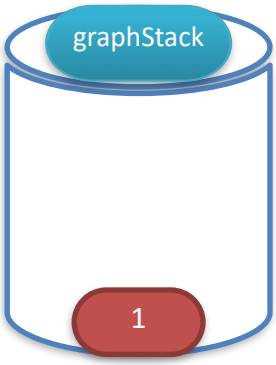
neighbour = 1{

Decrement by 1 the neighbours value in the inDeg.

inDeg ==> {0:-1, 1:0, 2:2, 3:3, 4:1, 5:1, 6:1} then,

0 = 0 true then push the neighbour in to the graphStack.

}



neighbour = 2{

Decrement by 1 the neighbours value in the inDeg.

inDeg ==> {0:-1, 1:0, 2:1, 3:3, 4:1, 5:1, 6:1} then,

1 = 0 false then do nothing.

}

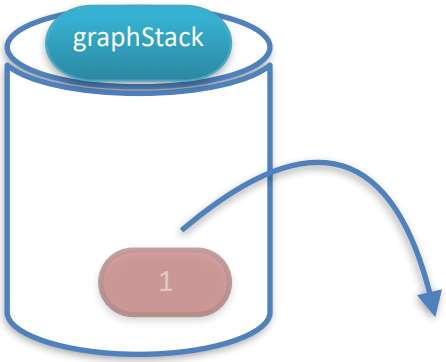
neighbour = 3{

Decrement by 1 the neighbours value in the inDeg.

inDeg ==> {0:-1, 1:0, 2:1, 3:2, 4:1, 5:1, 6:1} then,

2 = 0 false then do nothing.

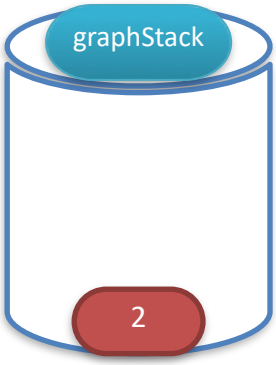
}



temp = 1

Then add to tem into the topSort.

topSort = {0,1}



neighbour = 2{

Decrement by 1 the neighbours value in the inDeg.

inDeg ==> {0:-1, 1:0, 2:0, 3:2, 4:1, 5:1, 6:1} then,

0 = 0 true then push the neighbour in to the graphStack.

}

neighbour = 3{

Decrement by 1 the neighbours value in the inDeg.

inDeg ==> {0:-1, 1:0, 2:0, 3:1, 4:1, 5:1, 6:1} then,

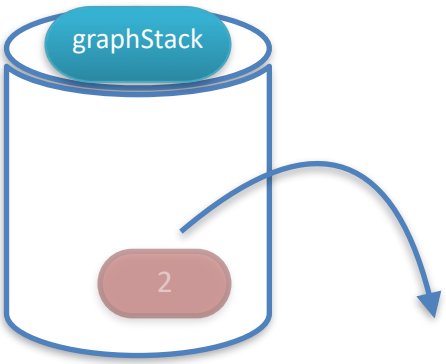
1 = 0 false then do nothing

}

temp = 2

Then add to tem into the topSort.

topSort = {0,1,2}



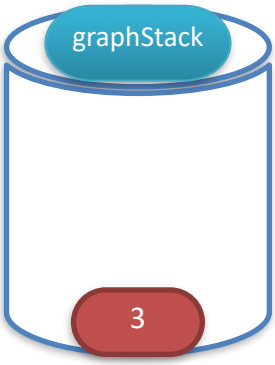
neighbour = 3{

Decrement by 1 the neighbours value in the inDeg.

inDeg ==> {0:-1, 1:0, 2:0, 3:0, 4:1, 5:1, 6:1} then,

0 = 0 true then push the neighbour in to the graphStack.

}



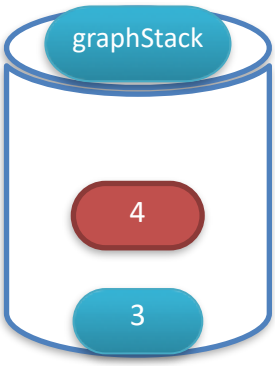
neighbour = 4{

Decrement by 1 the neighbours value in the inDeg.

inDeg ==> {0:-1, 1:0, 2:0, 3:0, 4:0, 5:1, 6:1} then,

0 = 0 false then push the neighbour in to the graphStack.

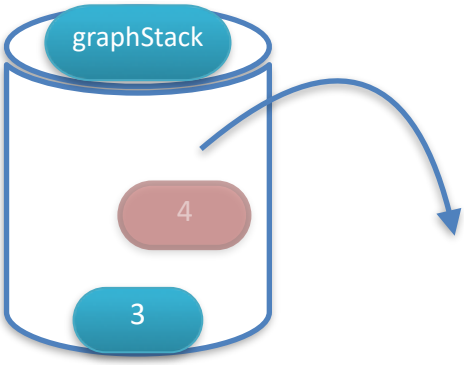
}



temp = 4

Then add to tem into the topSort.

topSort = {0,1,2,4}



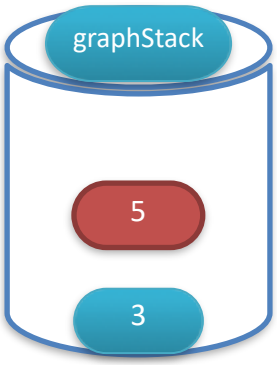
neighbour = 5{

Decrement by 1 the neighbours value in the inDeg.

inDeg ==> {0:-1, 1:0, 2:0, 3:0, 4:0, 5:0, 6:1} then,

0 = 0 false then push the neighbour in to the graphStack.

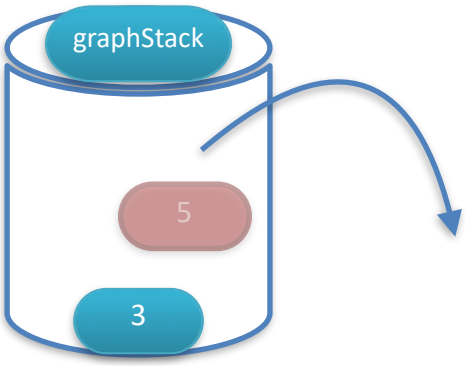
}



temp = 5

Then add to tem into the topSort.

topSort = {0,1,2,4,5}



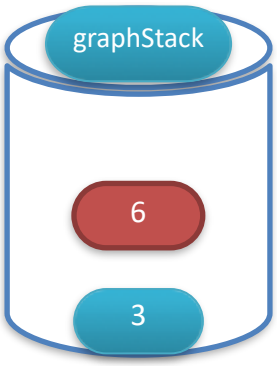
neighbour = 6{

Decrement by 1 the neighbours value in the inDeg.

inDeg ==> {0:-1, 1:0, 2:0, 3:0, 4:0, 5:0, 6:0} then,

0 = 0 false then push the neighbour in to the graphStack.

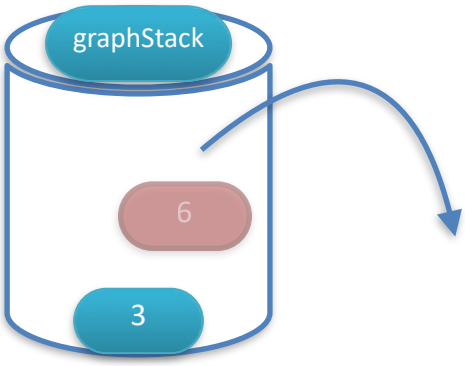
}



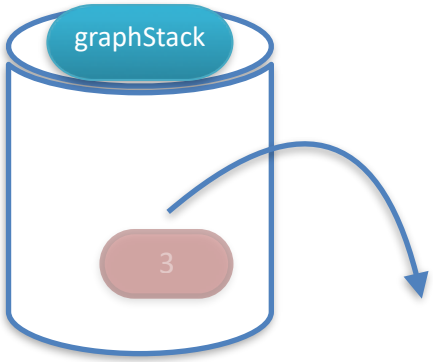
temp = 6

Then add to tem into the topSort.

topSort = {0,1,2,4,5,6}



There is no neighbour in the 6 vertex then,



temp = 3

Then add to tem into the topSort.

topSort = {0,1,2,4,5,6,3}

Then the graphStack is empty and break the while loops.

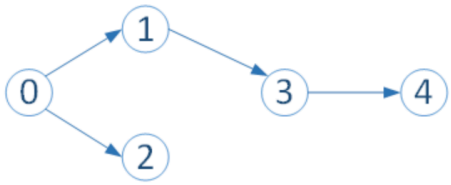
}

The topSort size = 7 and the adjList size = 7 the this is a directed acyclic graph that means there is no cycle in the graph.This algotrithm based on this idea if it is a dag there must be a vertex which has zero indegree and if we remove the vertex from the graph 1 by 1 (zero indegree vertex) then we have just 1 vertex. Instead of removing elements from the graph I just find the topological sort of the graph and if there is any loop the topological sort size doesn't equal to size of the adjList. My first submission is works but the result of the topological sort is wrong because of my bad implementation. When I trace the code I realised and I fixed it.

Challenge: Breadth First Search Distance

Time Complexity: O(V+E)

Example Input: 0 ,(0 1),(0 2),(1 3),(3 4)



Solution Code Trace:

```
53      public static Map<Integer, Integer> bfsDistance(DirectedGraph digraph, Integer start) {
54      Map<Integer, Integer> distance = new HashMap<Integer, Integer>();
55      for(Integer i: digraph.adjList.keySet()){
56          distance.put(i,-1);
57      }
58      distance.put(start,0);
59      Queue<Integer> digraphQueue = new LinkedList<Integer>();
60      digraphQueue.offer(start);
61      while (!digraphQueue.isEmpty()){
62          Integer temp = digraphQueue.poll();
63          int tempDist = distance.get(temp);
64          for(Integer neighbour: digraph.adjList.get(temp) ){
65              if (distance.get(neighbour) != -1) continue;
66              distance.put(neighbour,tempDist+1);
67              digraphQueue.offer(neighbour);
68          }
69      }
70          return distance;
71  }
```

distance ==> {0:-1, 1:-1, 2:-1, 3:-1, 4:-1} //firstly all of the distances are -1.

For start vertex the distance = 0 then after putting zero to value of the start vertex the distance look like this

distance ==> {0:0, 1:-1, 2:-1 ,3:-1, 4:-1}

Then I am adding the start vertex at the end of the digraphQueue.

digraphQueue ==>{0}

adjList ==> {0:[1,2], 1:[3], 2:[], 3:[4],4:[]}

digraphQueue is not empty{

temp = 0

digraphQueue ==>{}

tempDist = 0

neighbour = 1{

-1 != -1 false then , //checking the current neighbour has any distance.

Put the tempDist + 1 into the current neighbour's distance value.

distance ==> {0:0, 1:1, 2:-1, 3:-1, 4:-1}

And add the neighbour at the end of the digraphQueue.

digraphQueue ==>{1}

}

neighbour = 2{

-1 != -1 false then , //checking the current neighbour has any distance.

Put the tempDist + 1 into the current neighbour's distance value.

distance ==> {0:0, 1:1, 2:1, 3:-1, 4:-1}

And add the neighbour at the end of the digraphQueue.

digraphQueue ==>{1,2}

}

temp = 1

digraphQueue ==>{2}

tempDist = 1

neighbour =3{

-1 != -1 false then , //checking the current neighbour has any distance.

Put the tempDist + 1 into the current neighbour's distance value.

distance ==> {0:0, 1:1, 2:1, 3:2, 4:-1}

And add the neighbour at the end of the digraphQueue.

digraphQueue ==>{2,3}

}

temp = 2

digraphQueue ==>{3}

tempDist = 1

temp doesn't have any neighbour then,

temp = 3

digraphQueue ==>{}

tempDist = 2

neighbour =4{

-1 != -1 false then , //checking the current neighbour has any distance.

Put the tempDist + 1 into the current neighbour's distance value.

distance ==> {0:0, 1:1, 2:1, 3:2, 4:3}

And add the neighbour at the end of the digraphQueue.

digraphQueue ==>{4}

}

temp = 4

digraphQueue ==>{}

tempDist = 3

temp doesn't have any neighbour and digraphQueue is empty and break the while loops.

}

Return the distance ==> {0:0, 1:1, 2:1, 3:2, 4:3}

Default distance of the every vertex is -1 if a vertex has no way to reach start vertex then distance value of the vertex will be -1. And the other vertices which are have a way to reach start index.Their distance value will be the distance from the start vertex to the current vertex.

Challenge: Largest Permutation

Time Complexity: O(n)

Example Input: (5 1) , (4 2 3 5 1)

```
12  static int[] largestPermutation(int k, int[] arr) {
13     int len = arr.length;
14      int max= 0;
15      for (int i = 0; i <len &&k>0; i++) {
16          int j = 0;
17          while (j<len){
18              if (arr[j] == len-i){
19                  max = j;
20                  break;
21              }
22              j++;
23          }
24          if (i == max) continue;
25          int temp = arr[i];
26          arr[i] = arr[max];
27          arr[max] = temp;
28          k--;
29      }
30      return arr;
31  }
```

Challenge: Permuting Two Arrays

Time Complexity: O(n)

Example Input:

STDIN	Function
2	q = 2
3 10	A[] and B[] size n = 3, k = 10
2 1 3	A = [2, 1, 3]
7 8 9	B = [7, 8, 9]
4 5	A[] and B[] size n = 4, k = 5
1 2 2 1	A = [1, 2, 2, 1]
3 3 3 4	B = [3, 3, 3, 4]

```
12  static String twoArrays(int k, int[] A, int[] B) {
13          Arrays.sort(A);
14          Arrays.sort(B);
15          int[] reverseB = new int[B.length];
16          for (int i = B.length-1,j=0; i >=0&& j<B.length; i--,j++) {
17              reverseB[j] = B[i];
18          }
19
20
21
22          for (int i = 0; i <A.length ; i++) {
23              if (A[i] + reverseB[i]<k) return "NO";
24          }
25          return "YES";
26      }
```

Challenge: Beautiful Pairs

Time Complexity: O(n^2)

Example Input: 4,(1 2 3 4),(1 2 3 3)

```
    static int beautifulPairs(int[] A, int[] B) {
        Map<Integer,Integer> pairs = new HashMap<>();
        for (int i = 0; i <A.length ; i++) {
            for (int j = 0; j <B.length; j++) {
                if (A[i] == B[j]){
                    pairs.put(i,j);
                    A[i] = -1;
                    B[j] = -1;
                }
            }
        }
        if (pairs.size() ==A.length) return pairs.size()-1;
        return pairs.size()+1;
    }
```

Challenge: Luck Balance

Time Complexity: O(n)

Example Input:

```
6 3
5 1
2 1
1 1
8 1
10 0
5 0
```

$n = 6, k = 3$

$contests = [[5, 1], [2, 1], [1, 1], [8, 1], [10, 0], [5, 0]]$

```
    static int luckBalance(int k, int[][] contests) {
        List<Integer> impoCont = new ArrayList<>();
        int important = 0;
        int total= 0;
        for (int i = 0; i <contests.length ; i++) {
            total+= contests[i][0];
            if (contests[i][1] == 1){
                impoCont.add(contests[i][0]);
                important++;
            }
        }
        Collections.sort(impoCont);
        int diff = important-k;
        int lost = 0;
        while (diff>0){
            lost += impoCont.get(0);
            impoCont.remove(0);
            diff--;
        }
        return total-lost*2;

    }
```