```
title: "coursework3_nkatz01_dsa"
author: Nuchem Katz [redacted]
date: 09/01/2019



Question 1      Merging rows of a matrix

        import java.util.Arrays;

  public static void main(String[] args) {
    // Assumption that all cells are positive

    int[][] A = { // example
        { 1, 3, 5 }, { 1, 2, 6 }, { 4, 7, 8 }, };
    int cols = 3;
    int rows = 3;
    int minrow = 0;

    int[] Col0 = new int[rows];
    int[] PopOffset = new int[rows];
    int[] Sorted = new int[rows * cols];

    // Initialisation
    // copies A[i][0] to Col0
    for (int i = 0; i < rows; i++) {
      Col0[i] = A[i][0];
    }
    // puts zero in all PopOffset elements
    for (int j = 0; j < rows; j++) {
      PopOffset[j] = 0;
    }

    // main procedure
    // Generates sorted list lowest to highest in array Sorted[]
    for (int n = 0; n < (rows * cols); n++) { // this is n
      minrow = lowestRow(Col0); // this is n^2 */
      Sorted[n] = A[minrow][PopOffset[minrow]];
      PopOffset[minrow]++;
      if (PopOffset[minrow] <= cols - 1) {
        Col0[minrow] = A[minrow][PopOffset[minrow]];
      } else {
        Col0[minrow] = -1; /* Will be ignored */
      }

    }
    System.out.println(Arrays.toString(Sorted));

  }

  public static int lowestRow(int[] C) {
    // Procedure lowestRow returns the offset (index) of the lowest
    // positive value in the passed array. ie Ignore -1s in array
    int index = -1;
    int lowest = -1;
    for (int i = 0; i < C.length; i++) {
```

```
      if (C[i] != -1) {
        if (lowest == -1 || C[i] < lowest) {
          lowest = C[i];
          index = i;
        }
      }
    }
    return index;
  }
```

main procedure uses lowestRow(Col0) to find the lowest value in the first column in A,
then updating PopOffset with the next col index in A (and -1 if no more columns) of that row
from which the content was just popped, pointing that, together with the rowindex, back at A,
and copying the value this produces, into Col0, thus in effect, always having col0 as a
virtual mirror of the first col in A.

When an entire row in A is exhausted, a -1 is put in the cell in Col0, representing that row
and it is ignored by lowestRow().

Question 2 Processing elements of a binary tree

Call: System.out.println(Preorder(T)); from a main procedure.

```
public static int Preorder(T) //T points to the root of the tree
{
if(T!=null) //If T=null then we do nothing
{
int sum = 0;
if (T.content%2==0) {
          sum=sum+T.content;
}
sum =+ Preorder(T.LeftChild);
sum =+ Preorder(T.RightChild);
return sum;
}
}
```

Question 3 Binary search tree approximation

Call: System.out.println(NearestEl(T,x); from main procedure.

Public static double NearestEl(T,x)

```
                {

                If (T==null) { return 0; }       //T has no content
                If (T.content==x || (T.RightChild==null &&
T.LeftChild==null))  //T has neither children (or T is = x - maybe not
needed is assump is that x isn't in tree)
                return T.content;

                If (T.RightChild ==null && T.LeftChild !=null)  {
//T has only left child
                        if abs(x-NearestEl(T.LeftChild,x))>abs(x-
T.content);
                                Return T.content;
                        Else
                                Return NearestEl(T.LeftChild,x); }

                Else If (T.LeftChild==null && T.RightChild!=null) {
//T has only right child
                        if abs(x-NearestEl(T.RightChild,x))>abs(x-
T.content)
                                Return T.content;
                        Else
                                Return NearestEl(T.RightChild,x); }

                Else {                                                  //T
has both children


                        If abs(x-NearestEl(T.LeftChild,x))>abs(x-
T.content) {             //could be  T.Content or RightChild
                                if abs(x-
NearestEl(T.RightChild,x))>abs(x-T.content)

Return T.content;
                                                                        Else

Return NearestEl(T.RightChild,x);       }
                Else
//must be left child
                        Return NearestEl(T.LeftChild,x);

                }
                }
```

Question 5 Processing the list of edges

Question 5.1 Listing all the edges

```
         // first curly brackets within is row/vertice 0, second is
vertice 1 etc.
    int[][] A = { //example
      { 0, 1, 0, 0, 0, 1 },
      { 0, 0, 0, 0, 0, 1 },
      { 0, 0, 0, 1, 1, 0 },
      { 0, 0, 0, 0, 1, 0 },
      { 0, 0, 1, 0, 0, 0 },
      { 0, 0, 0, 0, 0, 0 },
      };

          public static void listingEdges(int[][] A) {
    for (int i = 0; i < A.length; i++) {
      for (int j = i + 1; j < A.length; j++) { // it's not printing
A{4,3} as there's no repetition

        if ((A[i][j] == 1))

          System.out.println((i) + " " + (j));
      }
    }
  }

        Call listingAllBridges(A);



        Question 5.2    Removal of an edge

                  public static void removalOfAnEdge(int[][] A, int r,
int c) {// receive index address to be ommitted when printing
                for (int i = 0; i < A.length; i++) {
                  for (int j = i + 1; j < A.length; j++) {// only
check the upper triangle in the matrix as the buttom triangle is

// just repitition
                        if (i == r && j == c)// This assures that G\e
is printed
                          continue;
                        if ((A[i][j] == 1) && (i != j))// no need to
print 'node adjacent to itself'
                          System.out.println((i) + " " + (j));
                  }
                }
          }


        Call removalOfAnEdge(A, r, c);// send index address to be
removed



        Question 5.3    Listing all the bridges

         public static void listingAllBridges(int[][] A) {
                for (int i = 0; i < A.length; i++) {
```

```
                    for (int j = i + 1; j < A.length; j++) {

                            if ((A[i][j] == 1)) {//only bother calling on
 connect, if there's an edge there
                              A[i][j] = 0;
                              if (connect(A) == false)//without this edge,
 the graph becomes disconnected
                               System.out.println((i) + " " + (j)); //this
 must be a bridge
                              A[i][j] = 1; //replace the edge regardless
                            }
                    }
                  }
            }

           Call listingAllBridges(A);
```

Question 4 Recognizing a star

Call System.out.println(RecogStar(A)); with A being the adjacency matrix that we would like to enquire.

```
public static boolean RecogStar(int[][] A) {
    boolean found = false;
    int v = 0;
    for (int j = 0; j < A.length; j++) {

      // checks if either 0 is the center vertex or any of the others
      if (((A[0][j] == 1) && (j == 0) && (A[0][j + 1] == 1)) || ((A[0][j] == 1)
&& !(j == 0))) {

        v = j;
        found = true;

        break;
      }

    }

    if (found) {

      for (int i = 0; i < A.length; i++) {
        // checks only the upper triangle of the array as the lower triangle is
just a
        // copy
        for (int j = i + 1; j < A.length; j++) {
```

```
          // checks first l to r, i.e. any column on the row = to v (for those
vertices
          // that fall in the upper triangle);//then vertically, any row on the
column =
          // to v (for those vertices that fall in the lower triangle - and so we
need to
          // treat the column index as our reference point in terms of what
vertex we're
          // at)
          if (((i == v) && (A[i][j] == 0)) || ((j == v) && (A[i][j] == 0))) {
            return false;

          }

          else {
            // if it's a 1 and neither of the vertices are v.
            if (A[i][j] == 1)

              if (!(i == v || j == v))
                return false;
              else
                continue;
          }

        }

      }

      return true;
    } else
      return false;

  }
```