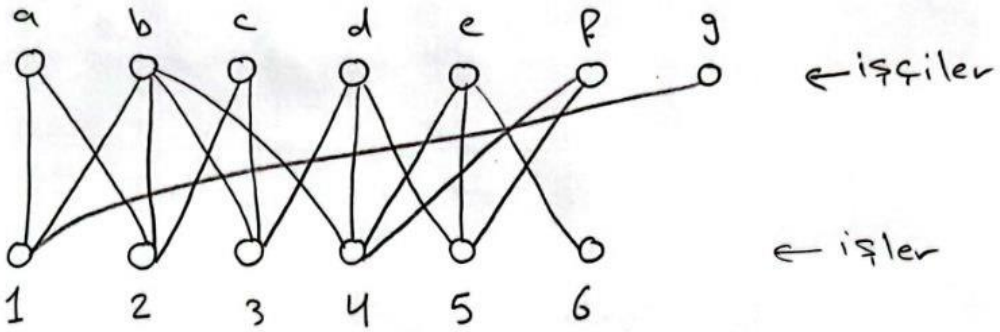
Malek Alismail / 20253833

Ödev sorusu 1:

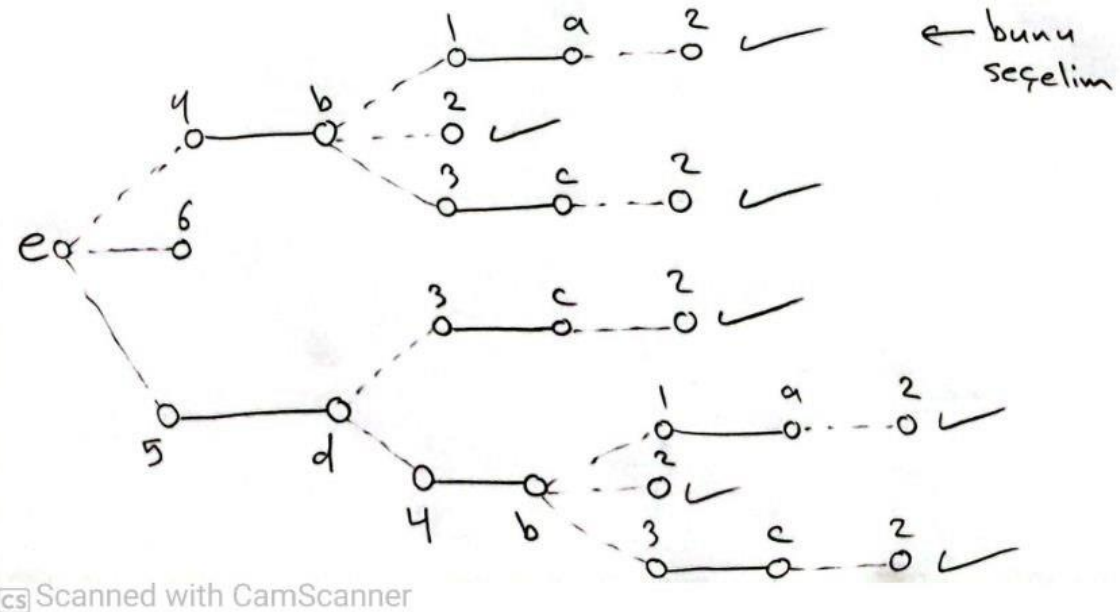
i) Bir işyerinde 6 işi yapabilecek 7 işçi bulunmaktadır. Hangi işçi hangi işi yapabileceğini gösteren graf aşağıda verilmiştir:



Çözüm:

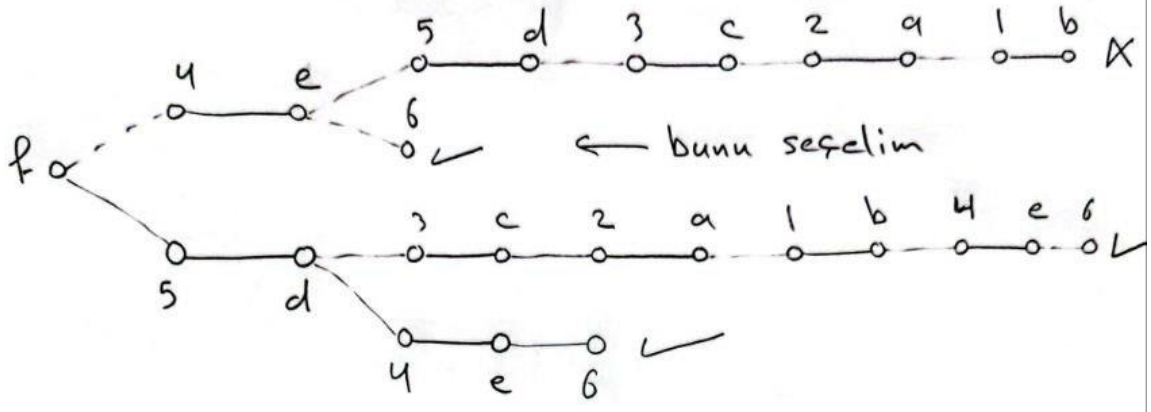
Başlangıç olarak $M = \{a_1, b_4, c_3, d_5\}$

- e'yi seçelim:



$M = \{e_4, b_1, a_2, c_3, d_5\}$

- f_4'i seçelim:



$M = \{a_2, b_1, c_3, d_5, e_6, f_4\}$ En büyük eşleme

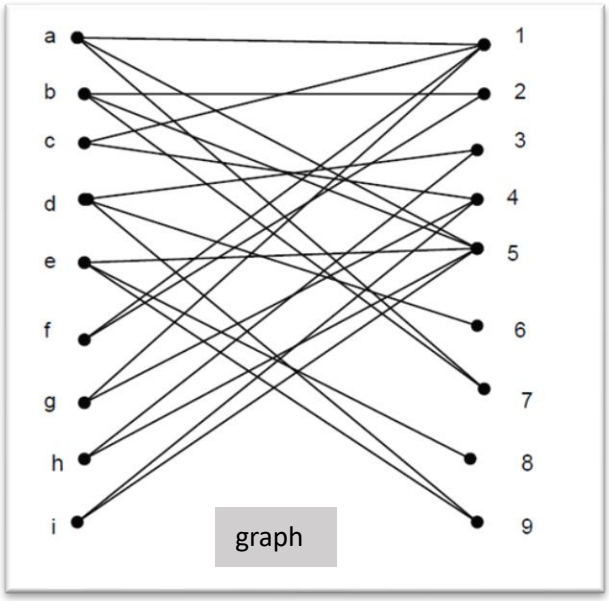
Mükemmel eşleme değildir.

Malek Alismail / 20253833

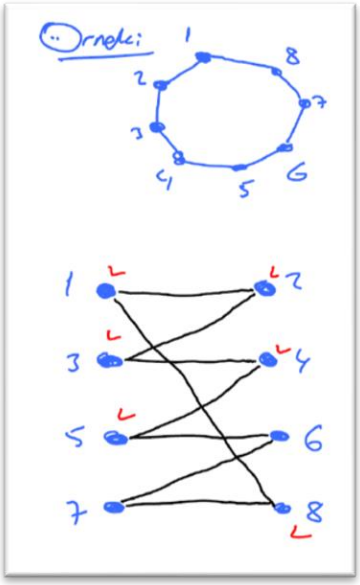
Ödev sorusu 1:

En büyük eşleme bulan java kodudur

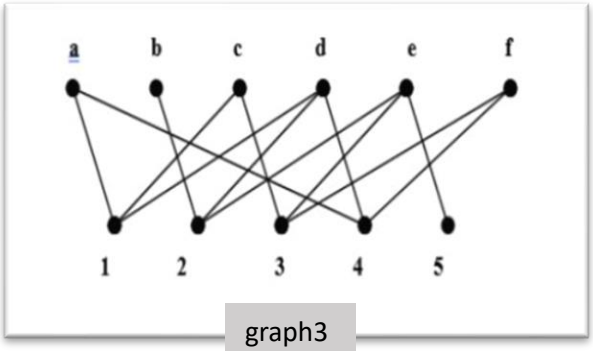
Gördüğümüz tüm örnekleri denedim + benden 2 tane örnek var



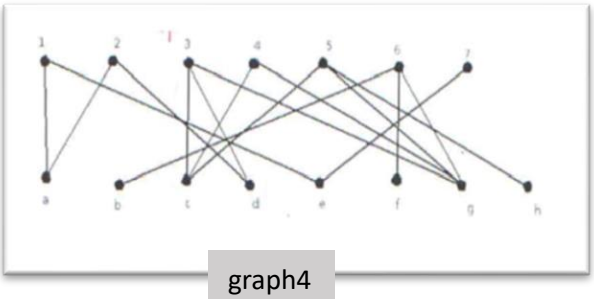
graph



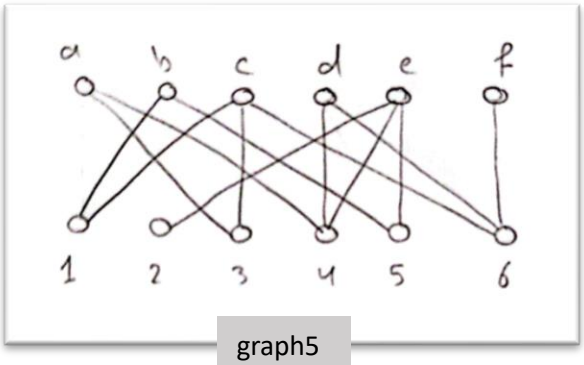
graph2



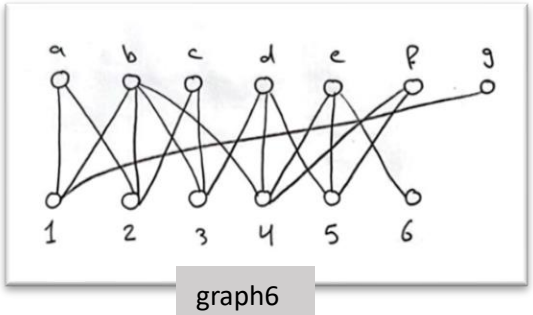
graph3



graph4



graph5



graph6

```java
import java.util.*;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.Set;
import java.util.HashSet;


public class Main {


    public static HashMap<String , Integer> firstM
            (int[][] graph , List<String> markedRows , List<Integer>
markedColumns ,
             String[] rows , int[] columns , Set<Integer> values)

    {
        HashMap<String , Integer> firstM = new HashMap<>();

        for (int i = 0; i < graph.length; i++) { //first M
            for (int j = 0; j < graph[0].length; j++) {
                if (graph[i][j] == 1 && !markedRows.contains(rows[i]) &&
!markedColumns.contains(columns[j])){

                    markedRows.add(rows[i]);
                    markedColumns.add(columns[j]);

                    firstM.put(rows[i] , columns[j]);

                    //the -1 is because the columnVertices starts from 1 and
in the array it starts from 0
                    values.add(columns[j] -1);
                }
            }
        }

        System.out.println("First M : " + firstM);

        return firstM;
    }


    public static List<String> allEdges
            (int[][] graph , String[] rows , int[] columns)
    {
        List<String> edges = new ArrayList<>();

        for (int i = 0; i < graph.length; i++) {
            for (int j = 0; j < graph[0].length; j++) {
                if (graph[i][j] == 1){
                    edges.add(rows[i] + columns[j]);
                }
            }
        }

        return edges;
    }
```

```
    public static void addLastEdge
            (int[][] graph , Set<Integer> values , HashMap<String , Integer>
M , int processedRow ,
             List<String> rows , int[] columns , Stack<String> path)

    {
        boolean bool = false;

        for (int i = 0; i < graph[0].length; i++) {
            if (graph[processedRow][i] == 1 && !values.contains(i)) {
                for (Map.Entry<String, Integer> entry : M.entrySet()) {
                    if (entry.getValue() != columns[i]) {
                        M.put(rows.get(processedRow), columns[i]);

                        path.add(rows.get(processedRow));
                        path.add(String.valueOf(columns[i]));
                        System.out.println("The path: " + path);
                        path.clear();

                        values.add(columns[i] - 1);
                        bool = true;
                        break;
                    }
                }
                if (bool) {
                    break;
                }
            }
        }
    }


    public static int findAugmentedPath
            (HashMap<String , Integer> M , HashMap<String , Integer> newM ,
HashMap<String , Integer> augmented ,
             List<String> rows , int[] columns , int processedRow , int i ,
Set<String> visitedRows , Stack<String> path ,
             Stack<Integer>[] visitedColumns)
    {

        for (Map.Entry<String, Integer> entry : M.entrySet()) {
            if (entry.getValue() == columns[i]) {
                visitedRows.add(rows.get(processedRow));

                for (int j = 0; j < rows.size(); j++) {
                    if (path.contains(rows.get(j))) {
                        continue;
                    }
                    visitedColumns[j].push(columns[i] - 1);
                }
                System.out.println("Visited rows: " + visitedRows);
                System.out.println("Vidited columns: \n"+
Arrays.toString(visitedColumns) + "\n");

                newM.put(rows.get(processedRow), columns[i]);
```

```
                augmented.put(entry.getKey(), entry.getValue());

                System.out.println(entry.getKey() + " is deleted");
                M.remove(entry.getKey());

                path.add(rows.get(processedRow));
                path.add(String.valueOf(columns[i]));

                processedRow = rows.indexOf(entry.getKey());

                System.out.println("M after deletion: " + M);
                System.out.println("newM: " + newM);
                System.out.println("The path: " + path);
                System.out.println("The new row to start with: " +
processedRow + "\n\n");
                break;
            }
        }

        return processedRow;
    }



    public static HashMap<String , Integer>  hungarian(int[][] graph){

        int n = graph.length; // rows count
        int m = graph[0].length; // columns count

        //to make sure that just one 1 is taken in first M from a certain row
or column
        List<String> markedRow = new ArrayList<>();
        List<Integer> markedColumn = new ArrayList<>();

        Set<Integer> values = new HashSet<>();

        String[] rowVertices = new String[n];
        int[] columnVertices = new int[m];

        String[] alphabet = {"a","b","c","d","e","f","g","h","i","j","k","l",
                "m","n","o","p","q","r","s","t","u","v","w","x","y","z"};

        System.arraycopy(alphabet, 0, rowVertices, 0, n); //giving the row
vertices names as letters

        for (int i = 0; i < m; i++) { //giving the column vertices names as
numbers
            columnVertices[i] = i+1;
        }




        HashMap<String , Integer> M = firstM(graph , markedRow , markedColumn
```

```
, rowVertices , columnVertices , values);
        List<String> allEdges = allEdges(graph , rowVertices ,
columnVertices); //to make sure that every edge is added


        int row,     //the row that is being processed
                min = Math.min(n, m), //if all the vertices from one side are
linked stop looping
                marked;
        boolean bool2 = false;

        //converting the rows to a list to make it easy to control a
contained vertex
        List<String> newRowVertices = new
ArrayList<>(Arrays.asList(rowVertices)),
                theOtherVertices = new ArrayList<>(newRowVertices); //the
vertices that aren't attached in the first M

        theOtherVertices.removeAll(markedRow);
        System.out.println("Vertices to match: "+theOtherVertices + "\n");

        /* for every vertex that isn't in the first M, the added edges in
        every loop are added to newM then the newM and M are merged */
        HashMap<String , Integer>  newM = new HashMap<>(),
                augmented = new HashMap<>();

        Stack<String> path = new Stack<>();
        Stack<Integer>[] visitedColumns = new Stack[newRowVertices.size()];
        Set<String> visitedRows = new HashSet<>();

        for (int i = 0; i < newRowVertices.size(); i++) {
            visitedColumns[i] = new Stack<>();
        }




        if (M.size() != min){
            for (String theOtherVertex : theOtherVertices) {
                if (M.size() == min){ //finish if one side is all linked
                    break;
                }
                row = newRowVertices.indexOf(theOtherVertex);

                System.out.println("\n***  The processed row vertex: {" +
                        theOtherVertex + "}, its row in the array: " + row +
"\n");

                for (int i = 0; i < m; i++) {
                    if (visitedColumns[row].contains(i)) {
                        continue;
                    }
                    if (graph[row][i] == 1) {
                        if (!values.contains(columnVertices[i] - 1)) { //if
the last edge of the augmented path is found
                            bool2 = true;
                            break;
```

```
                        }

                        row = findAugmentedPath(M , newM , augmented ,
newRowVertices , columnVertices ,
                                row , i , visitedRows , path ,
visitedColumns);

                        i = -1;

                    }

                    //if there is no augmented path try another path
                    if (i == m - 1 && !path.isEmpty()) {
                        for (int j = 0; j < newRowVertices.size(); j++) {
                            if (path.contains(newRowVertices.get(j))){
                                continue;
                            }
                            visitedColumns[j].pop();
                        }

                        System.out.println("The path: " + path + " *******");

                        marked = Integer.parseInt(path.pop()) - 1;

                        M.put(newRowVertices.get(row), marked + 1);

                        if (visitedRows.contains(newRowVertices.get(row)) &&
M.containsKey(newRowVertices.get(row))){
                            visitedColumns[row].pop();
                        }
                        visitedRows.remove(newRowVertices.get(row));

                        System.out.println("\nVisited rows: " + visitedRows);
                        System.out.println("Vidited columns: \n"+
Arrays.toString(visitedColumns) + "\n\n");

                        row = newRowVertices.indexOf(path.peek());
                        i = -1;
                        path.pop();
                    }
                }


                if (bool2) {

                    for (int i = 0; i < newRowVertices.size(); i++) {
                        visitedColumns[i].clear();
                    }
                    visitedRows.clear();
                    bool2 = false;

                    System.out.println("Adding the last edge in the augmented
path for the loop in vertex {"
                            + newRowVertices.get(row) + "}");

                    for (Map.Entry<String, Integer> entry : newM.entrySet())
{
```

```
                        if (!M.containsKey(entry.getKey())) {
                            // Only add the key-value pair if the key does
not already exist in M
                            M.put(entry.getKey(), entry.getValue());
                        }
                    }

                    newM.clear();
                    augmented.clear();

                    System.out.println("M after merging: " + M);

                    addLastEdge(graph , values , M , row , newRowVertices ,
columnVertices , path);

                    System.out.println("M after adding the last augmented
edge for vertex {" +
                            newRowVertices.get(row) + "} :" + M + "\n");

                } else {
                    M.putAll(augmented);
                    newM.clear();
                }
            }
        }


        M.putAll(newM);
        System.out.println("All edges: "+allEdges + "\n");

        if (m==n){
            if (min == M.size()){
                System.out.println("It is also Perfect matching.");
            }
        }



        return M;
    }




    public static void main(String[] args){
        int[][] graph = {
                {1, 0, 0, 0, 1, 0, 1, 0, 0},
                {0, 1, 0, 0, 1, 0, 1, 0, 0},
                {1, 0, 0, 1, 0, 0, 0, 0, 0},
                {0, 0, 1, 0, 0, 1, 0, 0, 1},
                {0, 0, 0, 0, 1, 0, 0, 1, 1},
                {1, 1, 0, 0, 0, 0, 0, 0, 0},
                {1, 0, 0, 1, 0, 0, 0, 0, 0},
                {0, 0, 1, 0, 1, 0, 0, 0, 0},
                {0, 0, 0, 1, 1, 0, 0, 0, 0}
        };
```

```
        int[][] graph2 = {
                {1, 0, 0, 1},
                {1, 1, 0, 0},
                {0, 1, 1, 0},
                {0, 0, 1, 1}
        };

        int[][] graph3 = {
                {1, 0, 0, 1, 0},
                {0, 1, 0, 0, 0},
                {1, 0, 1, 0, 0},
                {1, 1, 0, 1, 0},
                {0, 1, 1, 0, 1},
                {0, 0, 1, 1, 0}
        };

        int[][] graph4 = {
                {1, 1, 0, 0, 0, 0, 0},
                {0, 0, 0, 0, 0, 1, 0},
                {0, 0, 1, 1, 1, 0, 0},
                {0, 1, 1, 0, 0, 0, 0},
                {1, 0, 0, 0, 0, 0, 1},
                {0, 0, 0, 0, 0, 1, 0},
                {0, 0, 1, 1, 1, 1, 0},
                {0, 0, 0, 0, 1, 0, 0}
        };

        int[][] graph5 = {
                {0, 0, 1, 1, 0, 0},
                {1, 0, 0, 0, 1, 0},
                {1, 0, 1, 0, 0, 1},
                {0, 0, 0, 1, 0, 1},
                {0, 1, 0, 1, 1, 0},
                {0, 0, 0, 0, 0, 1}
        };

        int[][] graph6 = {
                {1, 1, 0, 0, 0, 0},
                {1, 1, 1, 1, 0, 0},
                {0, 1, 1, 0, 0, 0},
                {0, 0, 1, 1, 1, 0},
                {0, 0, 0, 1, 1, 1},
                {0, 0, 0, 1, 1, 0},
                {1, 0, 0, 0, 0, 0}
        };

        HashMap<String , Integer> result = hungarian(graph);
        System.out.println("The maximal matching: "+result);

    }
}
```

```java
public static void main(String[] args){
    int[][] graph = {
            {1, 0, 0, 0, 1, 0, 1, 0, 0},
            {0, 1, 0, 0, 1, 0, 1, 0, 0},
            {1, 0, 0, 1, 0, 0, 0, 0, 0},
            {0, 0, 1, 0, 0, 1, 0, 0, 1},
            {0, 0, 0, 0, 1, 0, 0, 1, 1},
            {1, 1, 0, 0, 0, 0, 0, 0, 0},
            {1, 0, 0, 1, 0, 0, 0, 0, 0},
            {0, 0, 1, 0, 1, 0, 0, 0, 0},
            {0, 0, 0, 1, 1, 0, 0, 0, 0}
    };
```

Main

```
The path: [i, 5] *******

Visited rows: [i]
Vidited columns:
[[], [4], [], [], [], [], [], [], [3, 4]]


All edges: [a1, a5, a7, b2, b5, b7, c1, c4, d3, d6, d9, e5, e8, e9, f1, f2, g1, g4, h3, h5, i4, i5]

The maximal matching: {a=7, b=5, c=4, d=6, e=8, f=2, g=1, h=3}

Process finished with exit code 0
```

```java
int[][] graph2 = {
        {1, 0, 0, 1},
        {1, 1, 0, 0},
        {0, 1, 1, 0},
        {0, 0, 1, 1}
};
```

Main

```
C:\Users\abobr\.jdks\openjdk-20\bin\java.exe "-javaagent:C:\Program Files\
First M : {a=1, b=2, c=3, d=4}
Vertices to match: []

All edges: [a1, a4, b1, b2, c2, c3, d3, d4]

It is also Perfect matching.
The maximal matching: {a=1, b=2, c=3, d=4}

Process finished with exit code 0
```

```java
int[][] graph3 = {
        {1, 0, 0, 1, 0},
        {0, 1, 0, 0, 0},
        {1, 0, 1, 0, 0},
        {1, 1, 0, 1, 0},
        {0, 1, 1, 0, 1},
        {0, 0, 1, 1, 0}
};
```

Main

```
C:\Users\abobr\.jdks\openjdk-20\bin\java.exe "-javaagent:C:\Program Files\JetBrains
First M : {a=1, b=2, c=3, d=4, e=5}
Vertices to match: [f]

All edges: [a1, a4, b2, c1, c3, d1, d2, d4, e2, e3, e5, f3, f4]

The maximal matching: {a=1, b=2, c=3, d=4, e=5}

Process finished with exit code 0
```

```java
int[][] graph4 = {
        {1, 1, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 1, 0},
        {0, 0, 1, 1, 1, 0, 0},
        {0, 1, 1, 0, 0, 0, 0},
        {1, 0, 0, 0, 0, 0, 1},
        {0, 0, 0, 0, 0, 1, 0},
        {0, 0, 1, 1, 1, 1, 0},
        {0, 0, 0, 0, 1, 0, 0}
};
```

Main

```
C:\Users\abobr\.jdks\openjdk-20\bin\java.exe "-javaagent:C:\Program Files\JetBrains\
First M : {a=1, b=6, c=3, d=2, e=7, g=4, h=5}
Vertices to match: [f]

All edges: [a1, a2, b6, c3, c4, c5, d2, d3, e1, e7, f6, g3, g4, g5, g6, h5]

The maximal matching: {a=1, b=6, c=3, d=2, e=7, g=4, h=5}

Process finished with exit code 0
```

```
int[][] graph5 = {
        {0, 0, 1, 1, 0, 0},
        {1, 0, 0, 0, 1, 0},
        {1, 0, 1, 0, 0, 1},
        {0, 0, 0, 1, 0, 1},
        {0, 1, 0, 1, 1, 0},
        {0, 0, 0, 0, 0, 1}
};
```

Main

```
Adding the last edge in the augmented path for the loop in vertex {b}
M after merging: {a=3, c=1, d=4, e=2, f=6}
The path: [f, 6, c, 1, b, 5]
M after adding the last augmented edge for vertex {b} :{a=3, b=5, c=1, d=4, e=2, f=6}

All edges: [a3, a4, b1, b5, c1, c3, c6, d4, d6, e2, e4, e5, f6]

It is also Perfect matching.
The maximal matching: {a=3, b=5, c=1, d=4, e=2, f=6}

Process finished with exit code 0
```

```
int[][] graph6 = {
        {1, 1, 0, 0, 0, 0},
        {1, 1, 1, 1, 0, 0},
        {0, 1, 1, 0, 0, 0},
        {0, 0, 1, 1, 1, 0},
        {0, 0, 0, 1, 1, 1},
        {0, 0, 0, 1, 1, 0},
        {1, 0, 0, 0, 0, 0}
};
```

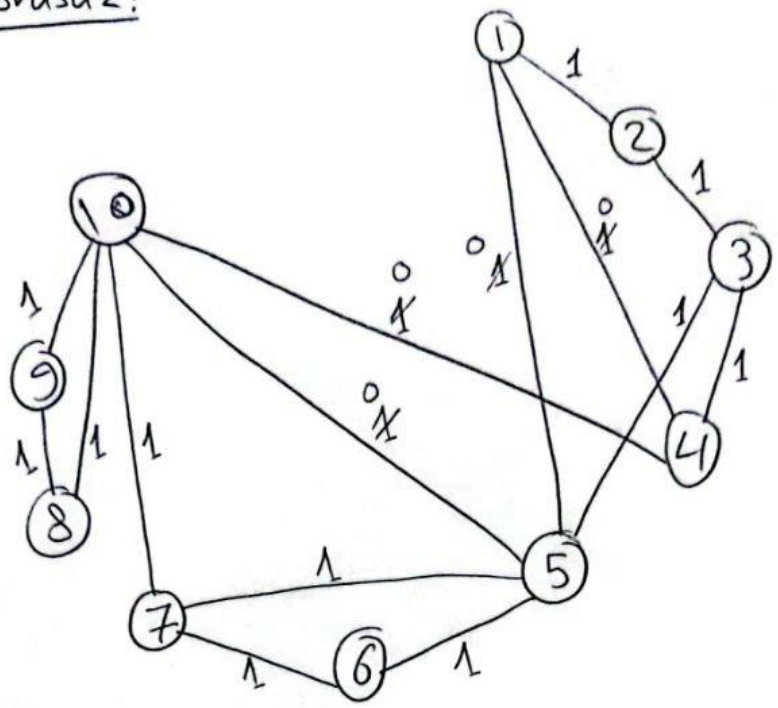
Main

```
Adding the last edge in the augmented path for the loop in vertex {e}
M after merging: {a=1, b=2, c=3, d=5, f=4}
The path: [f, 4, d, 5, e, 6]
M after adding the last augmented edge for vertex {e} :{a=1, b=2, c=3, d=5, e=6, f=4}

All edges: [a1, a2, b1, b2, b3, b4, c2, c3, d3, d4, d5, e4, e5, e6, f4, f5, g1]

The maximal matching: {a=1, b=2, c=3, d=5, e=6, f=4}

Process finished with exit code 0
```

Malek Alismail /2025 3833

Ödev sorusu2:

i)



1 ⟶ 10 En büyük akış:

- 1. yol {1-4-10} ⇒ min{1-4, 4-10} = 1 ⊗
- 2. yol {1-5-10} ⇒ min{1-5, 5-10} = 1 ⊗

içten ayrık yolların sayısı = 2 ⇒ $k(G) = 2$

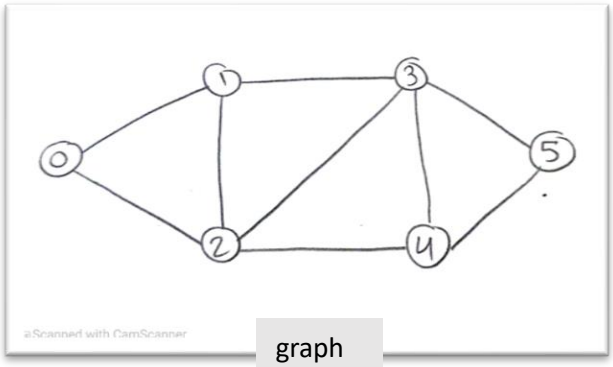
(4 ve 5 siliriz)

Ödev sorusu 2:

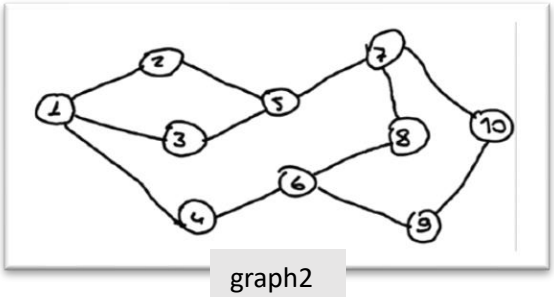
Connectivity number bulan bir java kodudur

Gördüğümüz tüm örnekleri denedim + benden 2 tane örnek var

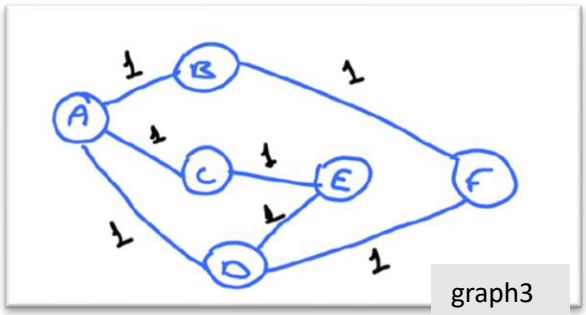
Note: source ve target tepeleri ekran çıktısında bulabilirsiniz.



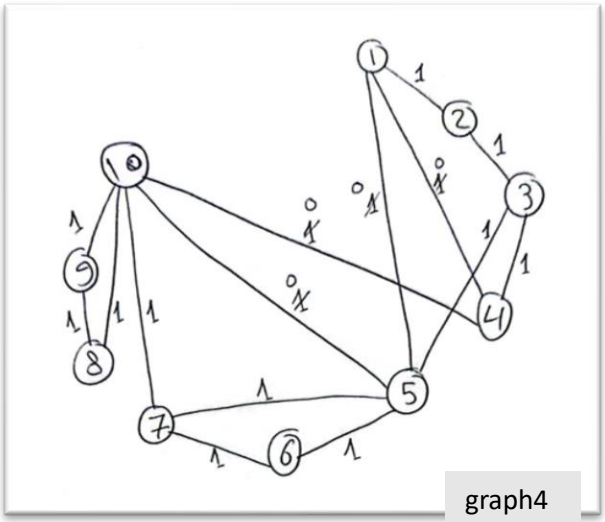
graph



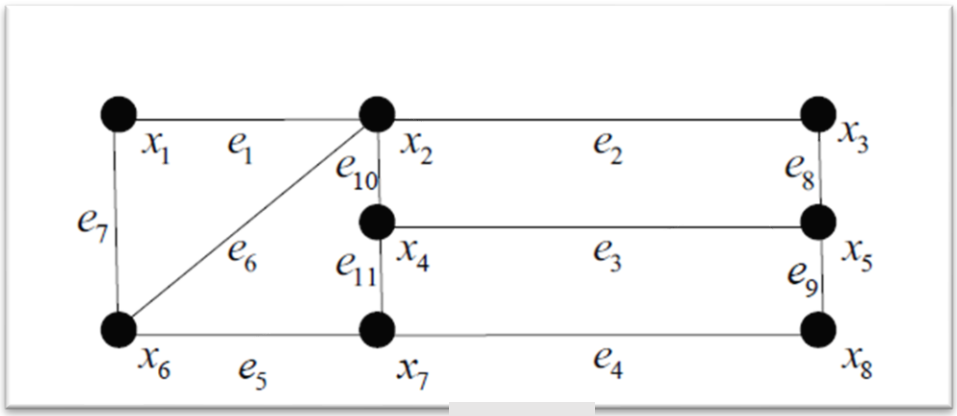
graph2



graph3



graph4



graph5

```java
import java.lang.*;
import java.util.*;

public class Main {
     public static List<Integer> writePath(int[] path , int pathNumber , int
source , int target){
         List<Integer> thePath = new LinkedList<>();
         thePath.add(target);

         for (int i = target ; i != source ; i=path[i]) {
             thePath.add(path[i]);
         }

         Collections.reverse(thePath);

         System.out.println((pathNumber) + ". Path: " + thePath + "\n");

         return thePath;
     }

     static Set<Integer> checkRepeated = new HashSet<>(); //check if the
vertex has been passed over

     public static boolean bfs(int[][] graph, int source, int target, int[]
parent , int pathNumber) {

        boolean[] visited = new boolean[graph.length]; //the visited vertices
in every path

        for (int i = 0; i < graph.length; ++i){
            visited[i] = false;
        }

        LinkedList<Integer> queue = new LinkedList<>();
        queue.add(source);
        visited[source] = true;

        List<Integer> passedVertices = new ArrayList<>(); //add all the
vertices that have been passed over from every path
        parent[source] = -1;

        while (queue.size() != 0) {
            int current = queue.poll();
            if (current != source && checkRepeated.contains(current)){
                continue;
            }

            for (int vertex = 0; vertex < graph.length; vertex++) {
                if (!visited[vertex] && graph[current][vertex] > 0) {

                    if (vertex == target) {
                        parent[vertex] = current;
                        passedVertices = writePath(parent , pathNumber ,
source , target);
                        checkRepeated.addAll(passedVertices);
                        return true;
                    }
```

```
                    queue.add(vertex);
                    parent[vertex] = current;
                    visited[vertex] = true;
                }
            }
        }
        return false;
    }

    public static int fordFulkerson(int[][] graph, int source, int target) {
        int u, v;

        int[] parent = new int[graph.length];

        int maxFlow = 0;

        int pathNumber = 1; //get the number of paths that bfs finds

        while (bfs(graph, source, target, parent , pathNumber)) {
            pathNumber++;

            int pathFlow = Integer.MAX_VALUE;

            for (v = target; v != source; v = parent[v]) { //get the minimum
flow of the selected path
                u = parent[v];
                pathFlow = Math.min(pathFlow, graph[u][v]);
            }

            for (v = target; v != source; v = parent[v]) {
                u = parent[v];
                graph[u][v] -= pathFlow;
                graph[v][u] += pathFlow;
            }

            maxFlow += pathFlow;
        }

        return maxFlow;
    }

    public static void main(String[] args){
        int[][] graph = {
                { 0, 1, 1, 0, 0, 0 },
                { 1, 0, 1, 1, 0, 0 },
                { 1, 1, 0, 1, 1, 0 },
                { 0, 1, 1, 0, 1, 1 },
                { 0, 0, 1, 1, 0, 1 },
                { 0, 0, 0, 1, 1, 0 }
        };

        //System.out.println("The connectivity number is " +
fordFulkerson(graph, 0, 5));

        int[][] graph2 = {
                { 0, 1, 1, 1, 0, 0, 0, 0, 0, 0},
                { 1, 0, 0, 0, 1, 0, 0, 0, 0, 0},
```

```
                { 1, 0, 0, 0, 1, 0, 0, 0, 0, 0},
                { 1, 0, 0, 0, 0, 1, 0, 0, 0, 0},
                { 0, 1, 1, 0, 0, 0, 1, 0, 0, 0},
                { 0, 0, 0, 1, 0, 0, 0, 1, 1, 0},
                { 0, 0, 0, 0, 1, 0, 0, 1, 0, 1},
                { 0, 0, 0, 0, 0, 1, 1, 0, 0, 0},
                { 0, 0, 0, 0, 0, 1, 0, 0, 0, 1},
                { 0, 0, 0, 0, 0, 0, 1, 0, 1, 0}
        };

        //System.out.println("The connectivity number is " +
fordFulkerson(graph2, 0, 9));

        int[][] graph3 = {
                { 0, 1, 1, 1, 0, 0 },
                { 1, 0, 0, 0, 0, 1 },
                { 1, 0, 0, 0, 1, 0 },
                { 1, 0, 0, 0, 1, 1 },
                { 0, 0, 1, 1, 0, 0 },
                { 0, 1, 0, 1, 0, 0 }
        };

        //System.out.println("The connectivity number is " +
fordFulkerson(graph3, 0, 5));

        int[][] graph4 = {
                { 0, 1, 0, 1, 1, 0, 0, 0, 0, 0},
                { 1, 0, 1, 0, 0, 0, 0, 0, 0, 0},
                { 0, 1, 0, 1, 1, 0, 0, 0, 0, 0},
                { 1, 0, 1, 0, 0, 0, 0, 0, 0, 1},
                { 1, 0, 1, 0, 0, 1, 1, 0, 0, 1},
                { 0, 0, 0, 0, 1, 0, 1, 0, 0, 0},
                { 0, 0, 0, 0, 1, 1, 0, 0, 0, 1},
                { 0, 0, 0, 0, 0, 0, 0, 0, 1, 1},
                { 0, 0, 0, 0, 0, 0, 0, 1, 0, 1},
                { 0, 0, 0, 1, 1, 0, 1, 1, 1, 0}
        };

        //System.out.println("The connectivity number is " +
fordFulkerson(graph4, 0, 9));

        int [][] graph5 = {
                {0, 1, 0, 0, 0, 1, 0, 0},
                {1, 0, 1, 1, 0, 1, 0, 0},
                {0, 1, 0, 0, 1, 0, 0, 0},
                {0, 1, 0, 0, 1, 0, 1, 0},
                {0, 0, 1, 1, 0, 0, 0, 1},
                {1, 1, 0, 0, 0, 0, 1, 0},
                {0, 0, 0, 1, 0, 1, 0, 1},
                {0, 0, 0, 0, 1, 0, 1, 0}
        };

        System.out.println("The connectivity number is " +
fordFulkerson(graph5, 4, 6));
    }
}
```

```java
no usages
public static void main(String[] args){
    int[][] graph = {
            { 0, 1, 1, 0, 0, 0 },
            { 1, 0, 1, 1, 0, 0 },
            { 1, 1, 0, 1, 1, 0 },
            { 0, 1, 1, 0, 1, 1 },
            { 0, 0, 1, 1, 0, 1 },
            { 0, 0, 0, 1, 1, 0 }
    };

    System.out.println("The connectivity number is " + fordFulkerson(graph, source: 0, target: 5));
```

```
Main
C:\Users\abobr\.jdks\openjdk-20\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community
1. Path: [0, 1, 3, 5]

2. Path: [0, 2, 4, 5]

The connectivity number is 2

Process finished with exit code 0
```

```java
    int[][] graph2 = {
            { 0, 1, 1, 1, 0, 0, 0, 0, 0, 0},
            { 1, 0, 0, 0, 1, 0, 0, 0, 0, 0},
            { 1, 0, 0, 0, 1, 0, 0, 0, 0, 0},
            { 1, 0, 0, 0, 0, 1, 0, 0, 0, 0},
            { 0, 1, 1, 0, 0, 0, 1, 0, 0, 0},
            { 0, 0, 0, 1, 0, 0, 0, 1, 1, 0},
            { 0, 0, 0, 0, 1, 0, 0, 1, 0, 1},
            { 0, 0, 0, 0, 0, 1, 1, 0, 0, 0},
            { 0, 0, 0, 0, 0, 1, 0, 0, 0, 1},
            { 0, 0, 0, 0, 0, 0, 1, 0, 1, 0}
    };

    System.out.println("The connectivity number is " + fordFulkerson(graph2, source: 0, target: 9));
```

```
Main
C:\Users\abobr\.jdks\openjdk-20\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community
1. Path: [0, 1, 4, 6, 9]

2. Path: [0, 3, 5, 8, 9]

The connectivity number is 2

Process finished with exit code 0
```

```java
int[][] graph3 = {
        { 0, 1, 1, 1, 0, 0 },
        { 1, 0, 0, 0, 0, 1 },
        { 1, 0, 0, 0, 1, 0 },
        { 1, 0, 0, 0, 1, 1 },
        { 0, 0, 1, 1, 0, 0 },
        { 0, 1, 0, 1, 0, 0 }
};

System.out.println("The connectivity number is " + fordFulkerson(graph3, source: 0, target: 5));
```

Main

```
C:\Users\abobr\.jdks\openjdk-20\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community
1. Path: [0, 1, 5]

2. Path: [0, 3, 5]

The connectivity number is 2

Process finished with exit code 0
```

```java
int[][] graph4 = {
        { 0, 1, 0, 1, 1, 0, 0, 0, 0, 0},
        { 1, 0, 1, 0, 0, 0, 0, 0, 0, 0},
        { 0, 1, 0, 1, 1, 0, 0, 0, 0, 0},
        { 1, 0, 1, 0, 0, 0, 0, 0, 0, 1},
        { 1, 0, 1, 0, 0, 1, 1, 0, 0, 1},
        { 0, 0, 0, 0, 1, 0, 1, 0, 0, 0},
        { 0, 0, 0, 0, 1, 1, 0, 0, 0, 1},
        { 0, 0, 0, 0, 0, 0, 0, 0, 1, 1},
        { 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1},
        { 0, 0, 0, 1, 1, 0, 1, 1, 1, 0}
};

System.out.println("The connectivity number is " + fordFulkerson(graph4, source: 0, target: 9));
```

Main

```
C:\Users\abobr\.jdks\openjdk-20\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community
1. Path: [0, 3, 9]

2. Path: [0, 4, 9]

The connectivity number is 2

Process finished with exit code 0
```

```java
    int [][] graph5 = {
            {0, 1, 0, 0, 0, 1, 0, 0},
            {1, 0, 1, 1, 0, 1, 0, 0},
            {0, 1, 0, 0, 1, 0, 0, 0},
            {0, 1, 0, 0, 1, 0, 1, 0},
            {0, 0, 1, 1, 0, 0, 0, 1},
            {1, 1, 0, 0, 0, 0, 1, 0},
            {0, 0, 0, 1, 0, 1, 0, 1},
            {0, 0, 0, 0, 1, 0, 1, 0}
    };

    System.out.println("The connectivity number is " + fordFulkerson(graph5,  source: 4,  target: 6));
}
```

Main

```
C:\Users\abobr\.jdks\openjdk-20\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community
1. Path: [4, 3, 6]

2. Path: [4, 7, 6]

3. Path: [4, 2, 1, 5, 6]

The connectivity number is 3
```