# Übungsblatt 10

*Truong, Debel*

## Aufgabe 2

a. Aufgabestellung: ungerichtete Graphen sind ausgewählt
→ Die Matrix ist immer symmetrisch durch $A_{i,j}$ wo $i = j$
Form:

| \ | i1 | i2 | i3 | i4 | i5 |
|---|---|---|---|---|---|
| 1j | 0 | A1 | A2 | A3 | A4 |
| 2j | A1 | 0 | B2 | B3 | B4 |
| 3j | A2 | B2 | 0 | C3 | C4 |
| 4j | A3 | B3 | C3 | 0 | D4 |
| 5j | A4 | B4 | C4 | D4 | 0 |

1. (4,3,3,2,2)

- deg(1) = 4, d.h (1) ist mit alle anderen Knoten verbindet

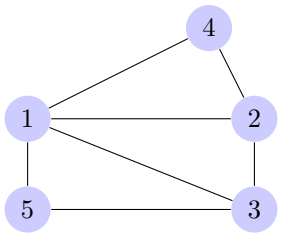
Adjazenzmatrix:

| \ | i1 | i2 | i3 | i4 | i5 | deg(n) |
|---|---|---|---|---|---|---|
| 1j | 0 | 1 | 1 | 1 | 1 | 4 |
| 2j | 1 | 0 | | | | 3 |
| 3j | 1 | | 0 | | | 3 |
| 4j | 1 | | | 0 | | 2 |
| 5j | 1 | | | | 0 | 2 |

$A_{1,1}$ ist definiert

deg(2) = deg(3) = 3 > deg'(2) = 2 d.h 2 von 3 leeren cell in 2 muss 1 sein > muss einmal in entweder i4 oder i5

| \ | i2 | i3 | i4 | i5 | deg'(n) |
|---|---|---|---|---|---|
| 2j | 0 | 1 | 1 | 0 | 2 |
| 3j | 1 | 0 | 0 | 1 | 2 |
| 4j | 1 | 0 | 0 | 0 | 1 |
| 5j | 0 | 1 | 0 | 0 | 1 |

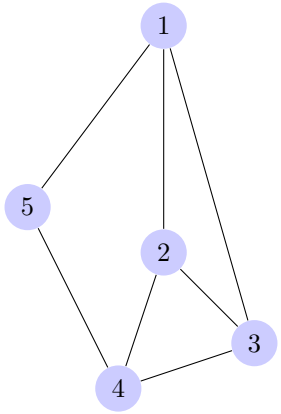



2. (5,3,2,2,2)

   $\Rightarrow$ ex. nicht, als deg(1) kann nur max. 4 sein ($A_{1,1}$ ist immer $0 \rightarrow V-1$ also 4 potentielle maximalle Verbindungen)

3. (3,3,3,3,2)

| \ | i1 | i2 | i3 | i4 | i5 | deg(n) |
|---|---|---|---|---|---|---|
| 1j | 0 | 1 | 1 | 0 | 1 | 3 |
| 2j | 1 | 0 | 1 | 1 | 0 | 3 |
| 3j | 1 | 1 | 0 | 1 | 0 | 3 |
| 4j | 0 | 1 | 1 | 0 | 1 | 3 |
| 5j | 1 | 0 | 0 | 1 | 0 | 2 |




4. (4,4,3,3,1)

   deg(1) = deg(2) = 4

| \ | i1 | i2 | i3 | i4 | i5 | deg(n) |
|---|---|---|---|---|---|---|
| 1j | 0 | 1 | 1 | 1 | 1 | 4 |
| 2j | 1 | 0 | 1 | 1 | 1 | 4 |
| 3j | 1 | 1 | 0 | | | 3 |
| 4j | 1 | 1 | | 0 | | 3 |
| 5j | 1 | 1 | | | 0 | 1 (!) |

$\Rightarrow$ ex. nicht

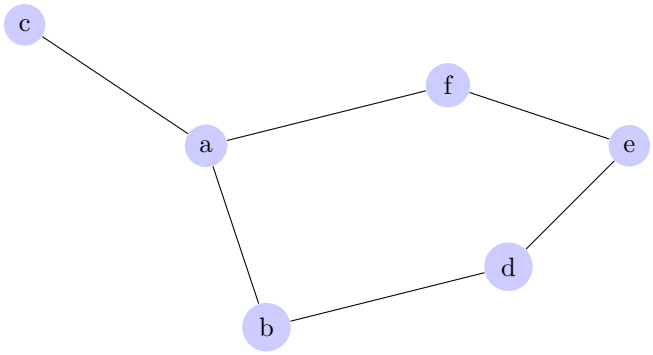
b.

- a,b,d,c,e,f (visual left to right, top to bottom)
- a,b,c,d,e,f (attempting left to right, top to bottom)

## Aufgabe 3

a. $G = (V, E)$ ungerichteter, zusammenhängender Graph

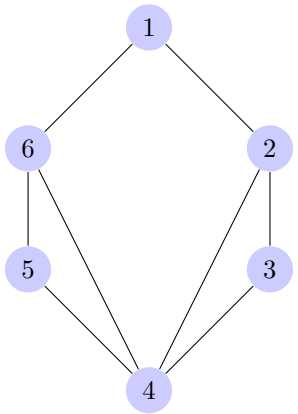
Example Graph as context:




Algorithmus: Näschste Seite

```
for (node : allNodes) {
  max = 0;
  deg = deg(node);
  if (deg > max) {
    max = deg;
    maxNode = selectNode(node);      //grabs the node with the max deg value which is a
  }
}

nodePool = allNodes;                //abcdef
arrayList travelList;
while (nodePool.hasNext()) {
  for (node : adjNodes(maxNode)) {   //pointer at a, adjNodes selects b,c,f
    if (deg(node) >= 2) {            //true cause deg(a) = 3, can also start with b,d,e,f
    arrayList traveseNodes = traverse(node);    //b,f,c, but will only add b,f later

    for (i : traverseNodes) {        //remove c, only keep b,f
      if (deg(i) < 2) {
        traverseNodes.remove(i);
      }
    }

    travelList.add(traverseNode);    //add b,f, travelList now a,b,f
    }
  nodePool.remove(node); //removes a, from poll, next iteration only checks b,d,e,f
  }
}                                    //final travelList: abd,bad,dbe,edf,fea

//dupeCount(travelList) counts the occurence of each repeating character
// in the (all characters in elements combined) List
// that means dupeCount(travellist) gives (3,3,3,3,3)

if (dupeCountEquals(travelList)) {   // dupeCount(travelList.find(a)) = dupeCount.find(b)
                                     // = ... = dupeCount.find(f) = 3
  return true;
} else return false;
```

b.




```
ArrayList pool;
```

```
for (node : Graph g) {
  pool.add(node);
}
//#1: select random edge
//let's say (1,2)
ArrayList randomEdge = pool.getRandom();
randomEdge.add(node.getRandomAdjNode());

//exclude from pool
// pool now (3,4,5,6)
pool.remove(randomEdge.getAll());

//select all adjNodes from #1
//we want to have 3,6 selected
ArrayList adjNodes = randomEdge.getAll().getAdjList(); //grabs (2,6) from node 1
                                                       //grabs (1,3) from node 2
for (node : randomEdge) {
  if (adjNodes.find(node) != null) { //finds 1 and 2 in adjNodes
    adjNodes.remove(node);
  }
} //only 3,6 left in adjNodes

//select all edges from adjNodes(1) exclude #1 //(6,5),(6,4)
//OR
//select all edges from adjNodes(2) exclude #1 //(3,4)
ArrayList matching;
for (node : adjNodes) { //for node 3 and 6

  //grabs (6,4),(6,5) from node 6
  //grabs (3,4) from node 3
  matching.add(node.getAdjList().remove(randomEdge.getAll()));
}

//now we exclude dupe in matching pool 5 - 6 - 4 - 3
//either removes (6,4) or (6,5),(3,4)
for (node : matching) { //delete until all degs equal 1
  if (deg(node) > 1) { //deg(6) = deg(4) = 2, deg(5) = 1
  matching.remove(node);
  }
} //only matching edges left

matching.add(randomEdge); //(1,2) + either (6,4) or (6,5),(3,4)
return matching;
```