Carson Sklare

2.

Code:

```
def undirect(d):
    adj = {}
    for u in d:
        adj[u] = []
    for u in d:
        for v in d[u]:
            adj[u].append(v)
            if v not in adj:
                adj[v] = []
            adj[v].append(u)
    return adj
```

Result with graph D

```
>>> D = {0: [11, 12], 1:[2,8,10,11], 2: [8,10,11], 3: [6, 7, 8], 4: [5,6], 5: [6],6:
[7,8], 7: [8], 8: [10,11], 9: [10], 10: [11], 11: [12]}
>>> undirect(D)
{0: [11, 12], 1: [2, 8, 10, 11], 2: [1, 8, 10, 11], 3: [6, 7, 8], 4: [5, 6], 5: [4,
6], 6: [3, 4, 5, 7, 8], 7: [3, 6, 8], 8: [1, 2, 3, 6, 7, 10, 11], 9: [10], 10: [1, 2
, 8, 9, 11], 11: [0, 1, 2, 8, 10, 12], 12: [0, 11]}
```

3.

A.

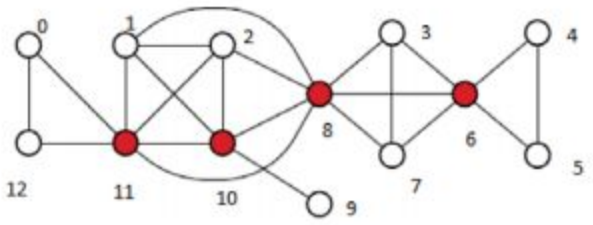
Code:

```
def is_cut(g,v):
    marked = {}
    nodes = {}
    count = 0
    for v2 in g:
        marked[v2] = False
        nodes[v2] = []
    for w in g[v]:
        if not marked[w]:
            nodes = is_cutr(g,w,marked,nodes)

    for i in range(len(nodes)):
        if nodes[i] == v:
            count+=1
    if count >= 2:
        return True
    return False

def is_cutr(g,v,marked,nodes):
    marked[v] = True
    for w in g[v]:
        if not marked[w]:
            nodes[w] = v
            is_cutr(g,w,marked,nodes)
    return nodes
```

Result:

```
>>> D = {0: [11, 12], 1:[2,8,10,11], 2: [8,10,11], 3: [6, 7, 8], 4: [5,6], 5: [6],6:
[7,8], 7: [8], 8: [10,11], 9: [10], 10: [11], 11: [12]}
>>> G = undirect(D)
>>> for v in G:
        print(v,is_cut(G,v),end = ',')


0 False,1 False,2 False,3 False,4 False,5 False,6 True,7 False,8 True,9 False,10 Tru
e,11 True,12 False,
```

B.
For reference graph D:



1. if v has at least two children in the DFS-tree rooted at v, then v is a cut-vertex
    a. **If you cut off a node with more than 2 children, for example take cutting off 11 in graph D, then children of node 11 that are not ancestors of 11 and not a direct descendant of 11 would not be able to reach each other. In the example 0 would not be able to reach 1 in the graph.**
2. if v has no child, or one child, in the DFS-tree rooted at v, then v is not a cut-vertex
    a. If you cut off a node with no or 1 child, the graph would still be connected because you will still be able to reach each node. For example, get rid of 0 who has a child 12 in D, every node including 12 will still be able to reach one another because they are connected to another node in some way.

4.
A.
Code:

```
def prereqs():
    D = {}
    while True:
        try:
            a,b = input('requires, required: ').split(',')
            a = a.strip()
            b = b.strip()
        except:
            break
        if a in D:
            D[a].append(b)
        else:
            D[a] = [b]
        if b not in D:
            D[b] = []
    return post_order(D)

def post_order(D):
    marked = {}
    prereq = []
    for v in D:
        marked[v] = False
    for v in D:
        if not marked[v]:
            post_orderR(D,v,marked,prereq)
    return prereq

def post_orderR(D,v,marked,prereq):
    marked[v] = True

    for w in D[v]:
        if not marked[w]:
            post_orderR(D,w,marked,prereq)
        if w not in prereq:
            prereq.append(w)
    if v not in prereq:
        prereq.append(v)
    return prereq
```

Result:

```
>>> prereqs()
requires, required: CSC 321, MAT 140
requires, required: MAT 141, MAT 140
requires, required: CSC 321, CSC 301
requires, required: CSC 301, CSC 300
requires, required: CSC 242, CSC 241
requires, required: CSC 300, CSC 241
requires, required:
['MAT 140', 'CSC 241', 'CSC 300', 'CSC 301', 'CSC 321', 'MAT 141', 'CSC 242']
```

B.
Code:

```
def prereqs2():
    D = {}
    while True:
        try:
            a,b = input('requires, required: ').split(',')
            a = a.strip()
            b = b.strip()
        except:
            break
        if a in D:
            D[a].append(b)
        else:
            D[a] = [b]
        if b not in D:
            D[b] = []
    if checkCycle(D):
        return post_order(D)
    else:
        return False

def checkCycle(D):
    new = []
    active = []
    finished = []
    for v in D:
        new.append(v)
    for v in D:
        if v in new:
            if checkCycleDFS(D,v,new,active,finished) == False:
                return False
    return True

def checkCycleDFS(D,v,new,active,finished):
    active.append(v)
    for w in D[v]:
        if w in active:
            return False
        elif w in new:
            if checkCycleDFS(D,w,new,active,finished) == False:
                return False
    active.remove(v)
    new.remove(v)
    finished.append(v)
    return True
```

Result:

```
>>> prereqs2()
requires, required: 1,2
requires, required: 2,3
requires, required: 1,4
requires, required:
['3', '2', '4', '1']
>>> prereqs2()
requires, required: 1,2
requires, required: 2,3
requires, required: 1,4
requires, required: 3,1
requires, required:
False
```