first name **last name** **id#**

each page 8 marks **30 min** **closed book** **no devices** **3 pages** **page 1**

1. In the box at right,
   for `H = [[0,1,2],[1,0,2],[1,0,2]]`
   and `R = [[1,2,0],[0,1,2],[2,1,0]]`,
   show the output printed by `m=propose_reject(H,R)`.

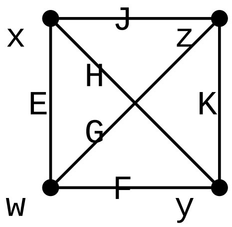
```
def propose_reject(H,R):
  n = pref_system_size(H,R)
  F,C = [None] * n, [0 for j in range(n)]
  rejection = True
  while rejection:
    rejection = False
    for j in range(n):
      h_choice = H[j][C[j]] # current H proposal
      if F[h_choice] == None: #R has no prop'ls
        F[h_choice] = j
        print(' ',j,' prop ',h_choice,': maybe')
      elif F[h_choice] != j: #R has 2 prop'ls
        r_maybe = F[h_choice] #R's current prop'l
        if prefers(R[h_choice], j, r_maybe):
          r_reject, r_maybe = r_maybe, j
          F[h_choice] = r_maybe
        else:
          r_reject = j
        print(' ',j,'prop',h_choice,
             ':pref',r_maybe,',rej',r_reject)
        C[r_reject] += 1 # H[j_rej.]: next pref
        rejection = True # a prop'l was rejected
  P = [H[j][C[j]] for j in range(n)]
  print('\nj  P  C  F')
  [print(j, P[j], C[j], F[j]) for j in range(n)]
  return P
```

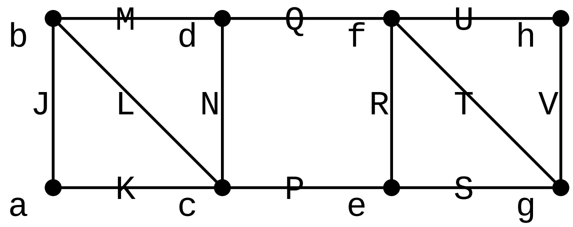
Show your rough work here.

2. Give a matching preference system with size 3 for which the propose-reject algorithm always finds a stable matching, or explain why this is not possible.

3.
```
def myunion(x,y,P):
  rootx = findGP(x,P)
  rooty = findGP(y,P)
  P[rootx] = rooty

def findGP(x, P):
  px = P[x]
  if x==px: return x
  gx = P[px] #grandparent
  while px != gx:
    P[x] = gx
    x = px
    px = gx
    gx = P[gx]
  return px
```

```
P represents 8 components of size 1:

j     0  1  2  3  4  5  6  7
P[j]  0  1  2  3  4  5  6  7

Now show P after myunion(2,3,P):

P[j] __ __ __ __ __ __ __ __

... and then after myunion(3,4,P):

P[j] __ __ __ __ __ __ __ __

... and then after myunion(4,5,P):

P[j] __ __ __ __ __ __ __ __

... and then after myunion(5,6,P):

P[j] __ __ __ __ __ __ __ __
```

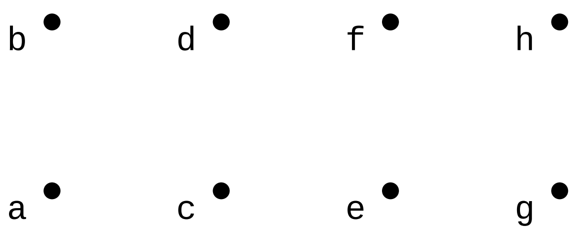




Recall: a *cut* of a graph is a partition of the node set into two non-empty subsets. E.g. on the small graph (above left), {{w,x}, {y,z}} is a cut with cross-edges {F,G,H,J}. RKMC is the randomized Kruskal min cut algorithm: unless otherwise stated, its input is a uniform-random permutation of the edges.

4. For the big graph, give each min cut (partition and cross-edges) ...

5. ... and give the forest (draw on the nodes below) and cut (partition and cross-edges) found by RKMC when edges are input in order LUQRKNTVSJMP.



6. Let G be a connected graph with a cut $\{X, Y\}$ with $G[X]$ (the subgraph of G on the node set X) connected but $G[Y]$ disconnected, with exactly two components $G[Y_1]$ and $G[Y_2]$. Prove or disprove: $\{X, Y\}$ is a min cut of G.

first name **last name** **id#**

each page 8 marks **30 min** **closed book** **no devices** **3 pages** **page 1**

1. In the box at right,
   for `H = [[1,2,0],[2,1,0],[2,1,0]]`
   and `R = [[0,1,2],[1,2,0],[0,2,1]]`,
   show the output printed by `m=propose_reject(H,R)`.

```
def propose_reject(H,R):
  n = pref_system_size(H,R)
  F,C = [None] * n, [0 for j in range(n)]
  rejection = True
  while rejection:
    rejection = False
    for j in range(n):
      h_choice = H[j][C[j]] # current H proposal
      if F[h_choice] == None: #R has no prop'ls
        F[h_choice] = j
        print(' ',j,' prop ',h_choice,': maybe')
      elif F[h_choice] != j: #R has 2 prop'ls
        r_maybe = F[h_choice] #R's current prop'l
        if prefers(R[h_choice], j, r_maybe):
          r_reject, r_maybe = r_maybe, j
          F[h_choice] = r_maybe
        else:
          r_reject = j
        print(' ',j,'prop',h_choice,
              ':pref',r_maybe,',rej',r_reject)
        C[r_reject] += 1 # H[j_rej.]: next pref
        rejection = True # a prop'l was rejected
  P = [H[j][C[j]] for j in range(n)]
  print('\nj  P  C  F')
  [print(j, P[j], C[j], F[j]) for j in range(n)]
  return P
```

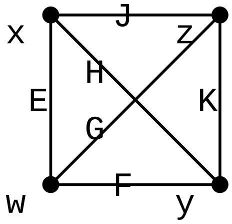
Show your rough work here.

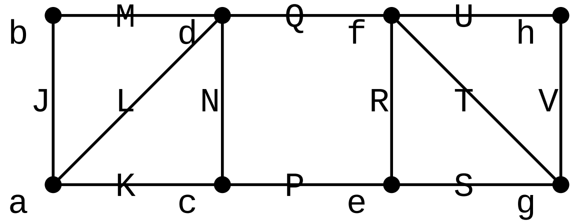
2. Give a matching preference system with size 3 for which the propose-reject algorithm always finds a stable matching, or explain why this is not possible.

3.
```
def myunion(x,y,P):
  rootx = findGP(x,P)
  rooty = findGP(y,P)
  P[rootx] = rooty

def findGP(x, P):
  px = P[x]
  if x==px: return x
  gx = P[px] #grandparent
  while px != gx:
    P[x] = gx
    x = px
    px = gx
    gx = P[gx]
  return px
```

```
Here P represents 8 components of size 1:

j     0  1  2  3  4  5  6  7
P[j]  0  1  2  3  4  5  6  7

Show P after myunion(3,4,P):

P[j] __ __ __ __ __ __ __ __

... and then after myunion(4,5,P):

P[j] __ __ __ __ __ __ __ __

... and then after myunion(5,6,P):

P[j] __ __ __ __ __ __ __ __

... and then after myunion(6,7,P):

P[j] __ __ __ __ __ __ __ __
```

first name **last name** **id#**

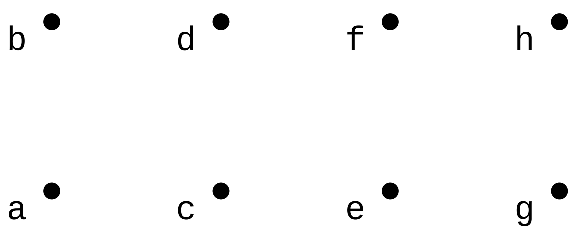




Recall: a *cut* of a graph is a partition of the node set into two non-empty subsets. E.g. on the small graph (above left), {{w,x}, {y,z}} is a cut with cross-edges {F,G,H,J}. RKMC is the randomized Kruskal min cut algorithm: unless otherwise stated, its input is a uniform-random permutation of the edges.

4. For the big graph, give each min cut (partition and cross-edges) . . .

5. . . . and give the forest (draw on the nodes below) and cut (partition and cross-edges) found by RKMC when edges are input in order QPRKNTMLJUVS.



6. Let G be a connected graph with a cut $\{X, Y\}$ with $G[X]$ (the subgraph of G on the node set X) connected but $G[Y]$ disconnected, with exactly two components $G[Y_1]$ and $G[Y_2]$. Prove or disprove: $\{X, Y\}$ is a min cut of G.

first name last name id#

each page 8 marks 30 min closed book no devices 3 pages

1. In the box at right,
   for `H = [[0,2,1],[2,0,1],[0,2,1]]`
   and `R = [[1,2,0],[0,2,1],[2,0,1]]`,
   show the output printed by `m=propose_reject(H,R)`.

```
def propose_reject(H,R):
  n = pref_system_size(H,R)
  F,C = [None] * n, [0 for j in range(n)]
  rejection = True
  while rejection:
    rejection = False
    for j in range(n):
      h_choice = H[j][C[j]] # current H proposal
      if F[h_choice] == None: #R has no prop'ls
        F[h_choice] = j
        print(' ',j,' prop ',h_choice,': maybe')
      elif F[h_choice] != j: #R has 2 prop'ls
        r_maybe = F[h_choice] #R's current prop'l
        if prefers(R[h_choice], j, r_maybe):
          r_reject, r_maybe = r_maybe, j
          F[h_choice] = r_maybe
        else:
          r_reject = j
        print(' ',j,'prop',h_choice,
              ':pref',r_maybe,',rej',r_reject)
        C[r_reject] += 1 # H[j_rej.]: next pref
        rejection = True # a prop'l was rejected
  P = [H[j][C[j]] for j in range(n)]
  print('\nj  P  C  F')
  [print(j, P[j], C[j], F[j]) for j in range(n)]
  return P
```

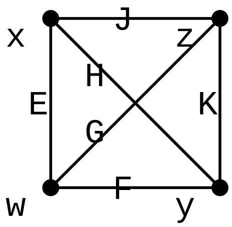
Show your rough work here.

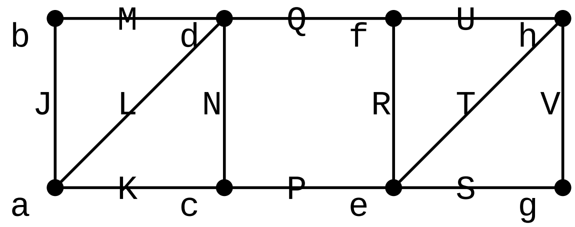
2. Give a matching preference system with size 3 for which the propose-reject algorithm always finds a stable matching, or explain why this is not possible.

3.

```
def myunion(x,y,P):
  rootx = findGP(x,P)
  rooty = findGP(y,P)
  P[rootx] = rooty

def findGP(x, P):
  px = P[x]
  if x==px: return x
  gx = P[px] #grandparent
  while px != gx:
    P[x] = gx
    x = px
    px = gx
    gx = P[gx]
  return px
```

```
Here P represents 8 components of size 1:

j     0  1  2  3  4  5  6  7
P[j]  0  1  2  3  4  5  6  7

Show P after myunion(1,2,P):

P[j] __ __ __ __ __ __ __ __

... and then after myunion(2,3,P):

P[j] __ __ __ __ __ __ __ __

... and then after myunion(3,4,P):

P[j] __ __ __ __ __ __ __ __

... and then after myunion(4,5,P):

P[j] __ __ __ __ __ __ __ __
```

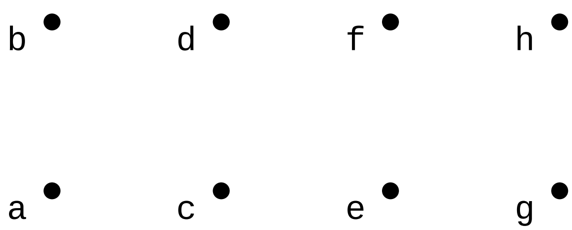




Recall: a *cut* of a graph is a partition of the node set into two non-empty subsets. E.g. on the small graph (above left), {{w,x}, {y,z}} is a cut with cross-edges {F,G,H,J}. RKMC is the randomized Kruskal min cut algorithm: unless otherwise stated, its input is a uniform-random permutation of the edges.

4. For the big graph, give each min cut (partition and cross-edges) …

5. …and give the forest (draw on the nodes below) and cut (partition and cross-edges) found by RKMC when edges are input in order SJTRVLNPUMKQ.



6. Let G be a connected graph with a cut $\{X, Y\}$ with $G[X]$ (the subgraph of G on the node set X) connected but $G[Y]$ disconnected, with exactly two components $G[Y_1]$ and $G[Y_2]$. Prove or disprove: $\{X, Y\}$ is a min cut of G.