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

each page 10 marks **30 min** **closed book** **no devices** **3 pages** **page 1**

1. At right, unscramble these lines from `kruskalDemo.py`. Write line numbers only: indent properly. We have written the first line number for you.

```
if ra != rb:                              #0   (2) ___ ___ ___ ___
for v in G: P[v] = v                      #1   ___ ___ ___ ___ ___
L = createEdgeList(G)                     #2   ___ ___ ___ ___ ___
P = {}                                    #3   ___ ___ ___ ___ ___
print(a,b,t[2])                           #4   ___ ___ ___ ___ ___
ra, rb = UF.myfind(a,P), UF.myfind(b,P)   #5   ___ ___ ___ ___ ___
t = extractmin(L)                         #6   ___ ___ ___ ___ ___
UF.myunion(ra,rb,P)                       #7   ___ ___ ___ ___ ___
while len(L) > 0:                         #8   ___ ___ ___ ___ ___
a, b = t[0], t[1]                         #9   ___ ___ ___ ___ ___
```

2. Here is the start of the analysis of extract-min from Kruskal's algorithm: `When we create the edge list` L`, we construct` L `from` G `which has` n `vertices. We have a for loop that iterates through all vertices` $v \in G$ `which takes` $O(n)$ `time. Then we iterate through the edges of` v`, and there could be` $n-1$ `of these, so we are now taking at least` $O(n^2)$ `time. The rest of the steps take constant time, so creating` L `takes` $O(n^2)$ `time. Next we begin a while loop that iterates through` L`,` L `has` $O(n^2)$ `edges so this now takes at least` $O(n^2)$ `time. Then extractmin iterates again through` L`, taking` $O(n^2)$ `time. In total, the algorithm takes` $O(n^4)$ `time. The rest of the steps in this while loop are dominated in time complexity from extractmin so, so far, total runtime is in` $O(n^4)$`.`

   (a) What should the final big O bound on the runtime be?

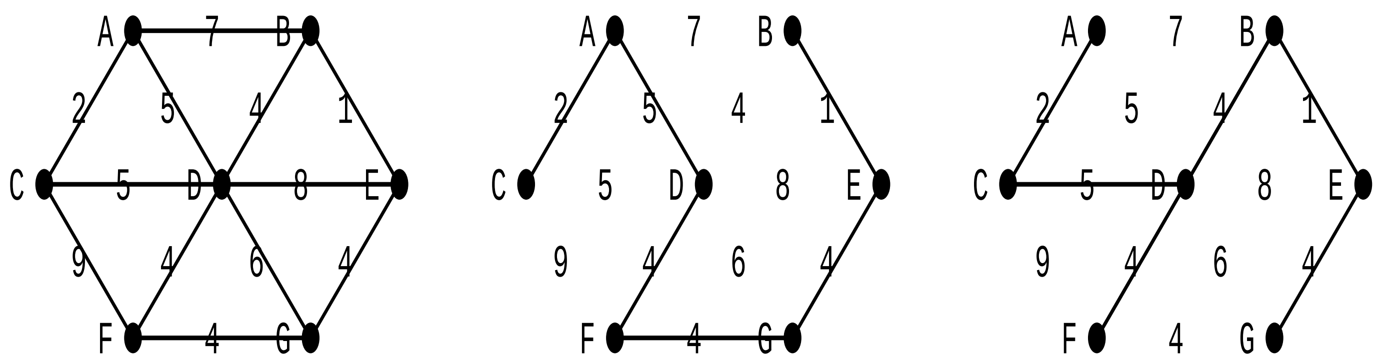
   (b) Finish the analysis.

3. Here is the start of a proof of correctness of Kruskal's algorithm. Input: a graph G with n nodes, output: a set K of edges of G. We want to show that K is an MST. Let M be any MST of G. Case 1: if $K = M$ then we are done. Case 2: assume K is not equal to M. Label edges $e_1, e_2, \ldots, e_m$ in the order considered by the algorithm. Let k be the smallest index such that e_k is not in M. Let C be the cycle of $M + e_k$. Let e_j be any edge of C that is not in K. Let M_1 be $M + e_k - e_j$. M_1 is a spanning subgraph of G. M_1 is connected and so a spanning tree. Cost(e_j) $\geq$ cost(e_k) ($*$). Cost(e_j) $\leq$ cost(e_k) ($**$). M_1 is an MST ($***$).

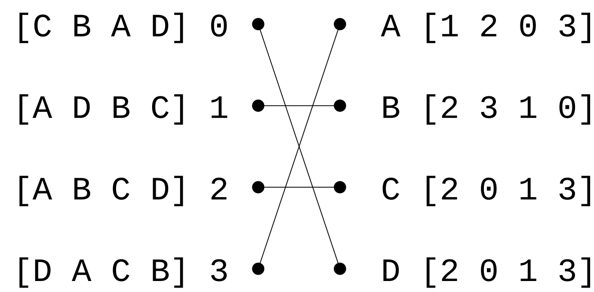
   (a) Justify ($*$)

   (b) Finish the proof.

4. Below is a graph G, the edge set K returned by Kruskal's algorithm, and an MST M. In the above proof, what is e_k? What are the possible choices for e_j?

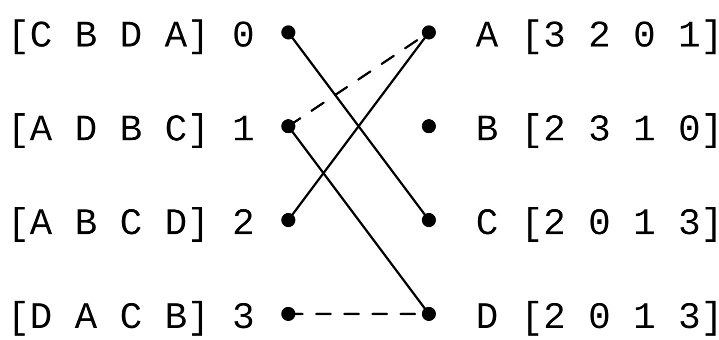


5. Is {0,A} an unhappy couple? Explain carefully.



Is {3,D} an unhappy couple? Explain carefully.

6. Here is a diagram of the propose-maybe-reject stable matching algorithm after some number of rounds.



a) For each proposer (hospital), in the next round, what new proposals are made? Explain carefully.

b) After the new proposals are made, what rejections are made by maybe-rejecters (residents)? Explain carefully.

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

each page 10 marks **30 min** **closed book** **no devices** **3 pages** **page 1**

1. At right, unscramble these lines from `kruskalDemo.py`. Write line numbers only: indent properly. We have written the first line number for you.

```
while len(L) > 0:                            #0  (4) ___ ___ ___ ___
a, b = t[0], t[1]                            #1  ___ ___ ___ ___ ___
if ra != rb:                                 #2  ___ ___ ___ ___ ___
for v in G: P[v] = v                         #3  ___ ___ ___ ___ ___
L = createEdgeList(G)                        #4  ___ ___ ___ ___ ___
P = {}                                       #5  ___ ___ ___ ___ ___
print(a,b,t[2])                              #6  ___ ___ ___ ___ ___
ra, rb = UF.myfind(a,P), UF.myfind(b,P)      #7  ___ ___ ___ ___ ___
t = extractmin(L)                            #8  ___ ___ ___ ___ ___
UF.myunion(ra,rb,P)                          #9  ___ ___ ___ ___ ___
```

2. Here is the start of the analysis of extract-min from Kruskal's algorithm: `When we create the edge list` L`, we construct` L `from` G `which has` n `vertices. We have a for loop that iterates through all vertices` $v \in G$ `which takes` $O(n)$ `time. Then we iterate through the edges of` v`, and there could be` $n-1$ `of these, so we are now taking at least` $O(n^2)$ `time. The rest of the steps take constant time, so creating` L `takes` $O(n^2)$ `time. Next we begin a while loop that iterates through` L`,` L `has` $O(n^2)$ `edges so this now takes at least` $O(n^2)$ `time. Then extractmin iterates again through` L`, taking` $O(n^2)$ `time. In total, the algorithm takes` $O(n^4)$ `time. The rest of the steps in this while loop are dominated in time complexity from extractmin so, so far, total runtime is in` $O(n^4)$`.`

   (a) What should the final big O bound on the runtime be?

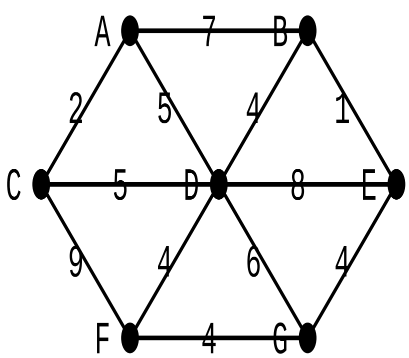
   (b) Finish the analysis.

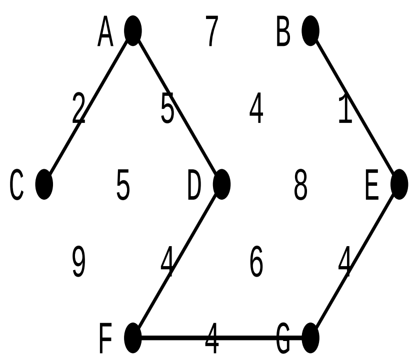
3. Here is the start of a proof of correctness of Kruskal's algorithm. Input: a graph G with n nodes, output: a set K of edges of G. We want to show that K is an MST. Let M be any MST of G. Case 1: if $K = M$ then we are done. Case 2: assume K is not equal to M. Label edges $e_1, e_2, \ldots, e_m$ in the order considered by the algorithm. Let k be the smallest index such that e_k is not in M. Let C be the cycle of $M + e_k$. Let e_j be any edge of C that is not in K. Let M_1 be $M + e_k - e_j$. M_1 is a spanning subgraph of G. M_1 is connected and so a spanning tree. $\text{Cost}(e_j) \geq \text{cost}(e_k)$ $(*)$. $\text{Cost}(e_j) \leq \text{cost}(e_k)$ $(**)$. M_1 is an MST $(***)$.

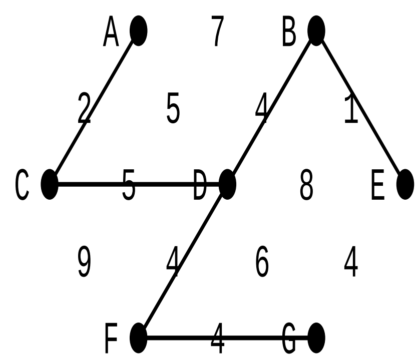
   (a) Justify $(**)$

   (b) Finish the proof.

4. Below is a graph G, the edge set K returned by Kruskal's algorithm, and an MST M. In the above proof, what is e_k? What are the possible choices for e_j?

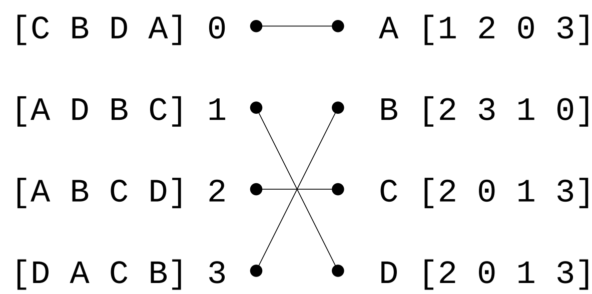




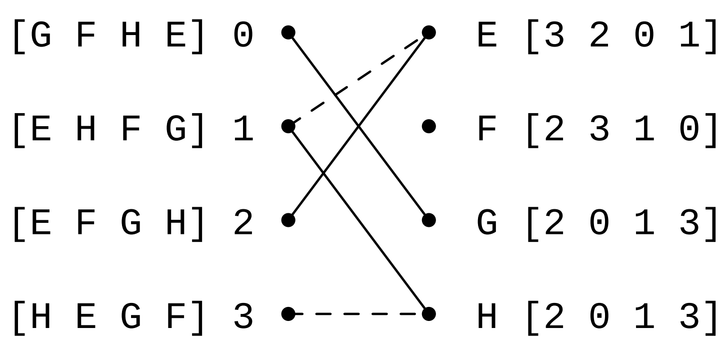


5. Below, is {2,B} an unhappy couple? Explain carefully.

   Below, is {1,C} an unhappy couple? Explain carefully.



6. Here is a diagram of the propose-maybe-reject stable matching algorithm after some number of rounds.



   a) For each proposer (hospital), in the next round, what new proposals are made? Explain carefully.

   b) After the new proposals are made, what rejections are made by maybe-rejecters (residents)? Explain carefully.

1. At right, unscramble these lines from `kruskalDemo.py`. Write line numbers only: indent properly. We have written the first line number for you.

```
P = {}                                      #0   (9_ ___ ___ ___ ___
print(a,b,t[2])                             #1   ___ ___ ___ ___ ___
ra, rb = UF.myfind(a,P), UF.myfind(b,P)     #2   ___ ___ ___ ___ ___
t = extractmin(L)                           #3   ___ ___ ___ ___ ___
UF.myunion(ra,rb,P)                         #4   ___ ___ ___ ___ ___
while len(L) > 0:                           #5   ___ ___ ___ ___ ___
a, b = t[0], t[1]                           #6   ___ ___ ___ ___ ___
if ra != rb:                                #7   ___ ___ ___ ___ ___
for v in G: P[v] = v                        #8   ___ ___ ___ ___ ___
L = createEdgeList(G)                       #9   ___ ___ ___ ___ ___
```

2. Here is the start of the analysis of extract-min from Kruskal's algorithm: `When we create the edge list` L`, we construct` L `from` G `which has` n `vertices. We have a for loop that iterates through all vertices` $v \in G$ `which takes` $O(n)$ `time. Then we iterate through the edges of` v`, and there could be` $n-1$ `of these, so we are now taking at least` $O(n^2)$ `time. The rest of the steps take constant time, so creating` L `takes` $O(n^2)$ `time. Next we begin a while loop that iterates through` L`,` L `has` $O(n^2)$ `edges so this now takes at least` $O(n^2)$ `time. Then extractmin iterates again through` L`, taking` $O(n^2)$ `time. In total, the algorithm takes` $O(n^4)$ `time. The rest of the steps in this while loop are dominated in time complexity from extractmin so, so far, total runtime is in` $O(n^4)$`.`

   (a) What should the final big O bound on the runtime be?

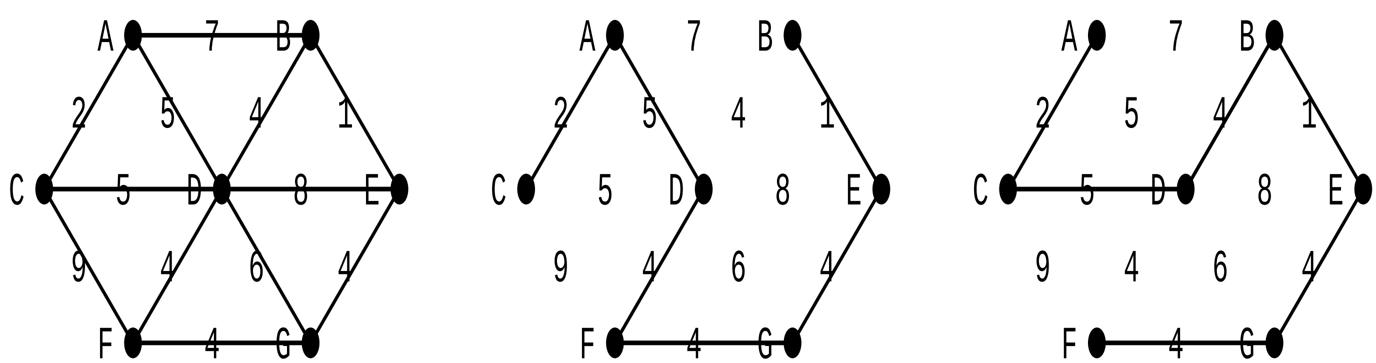
   (b) Finish the analysis.

3. Here is the start of a proof of correctness of Kruskal's algorithm. Input: a graph G with n nodes, output: a set K of edges of G. We want to show that K is an MST. Let M be any MST of G. Case 1: if $K = M$ then we are done. Case 2: assume K is not equal to M. Label edges $e_1, e_2, \ldots, e_m$ in the order considered by the algorithm. Let k be the smallest index such that e_k is not in M. Let C be the cycle of $M + e_k$. Let e_j be any edge of C that is not in K. Let M_1 be $M + e_k - e_j$. M_1 is a spanning subgraph of G. M_1 is connected and so a spanning tree. Cost$(e_j) \geq$ cost(e_k) $(*)$. Cost$(e_j) \leq$ cost(e_k) $(**)$. M_1 is an MST $(***)$.

   (a) Justify $(***)$

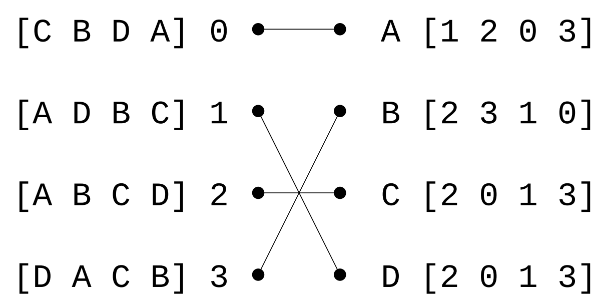
   (b) Finish the proof.

4. Below is a graph G, the edge set K returned by Kruskal's algorithm, and an MST M. In the above proof, what is e_k? What are the possible choices for e_j?

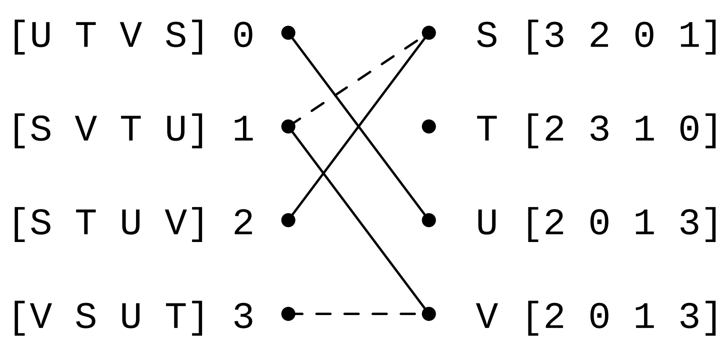


5. Below, is {0,B} an unhappy couple? Explain carefully.

   Below, is {3,C} an unhappy couple? Explain carefully.



6. Here is a diagram of the propose-maybe-reject stable matching algorithm after some number of rounds.



   a) For each proposer (hospital), in the next round, what new proposals are made? Explain carefully.

   b) After the new proposals are made, what rejections are made by maybe-rejecters (residents)? Explain carefully.