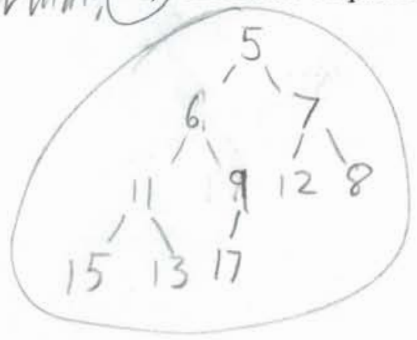
Key

1)
X)

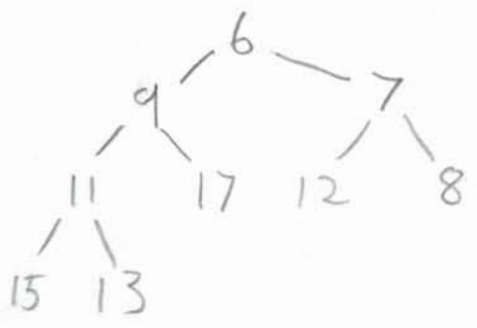
Starting with an empty binary min-heap:

linear? normal?

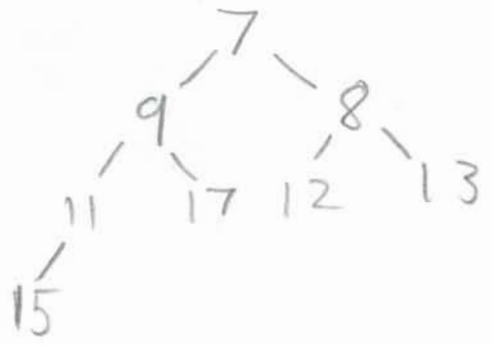
a) Insert the sequence 5, 9, 7, 11, 17, 12, 8, 15, 13, 6. Draw the final binary min-heap.



b) Perform a deleteMin on your binary min-heap. Draw the resulting binary min-heap.



c) Perform a deleteMin on your binary min-heap. Draw the resulting binary min-heap.



0 or 1 as root?

d) Draw an array representation of your binary min-heap from c).

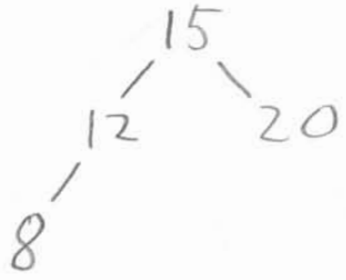
X	7	9	8	11	17	12	13	15

2

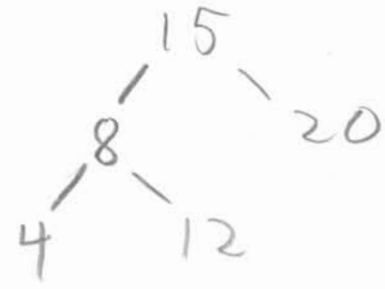
X)

Starting with an empty AVL tree:

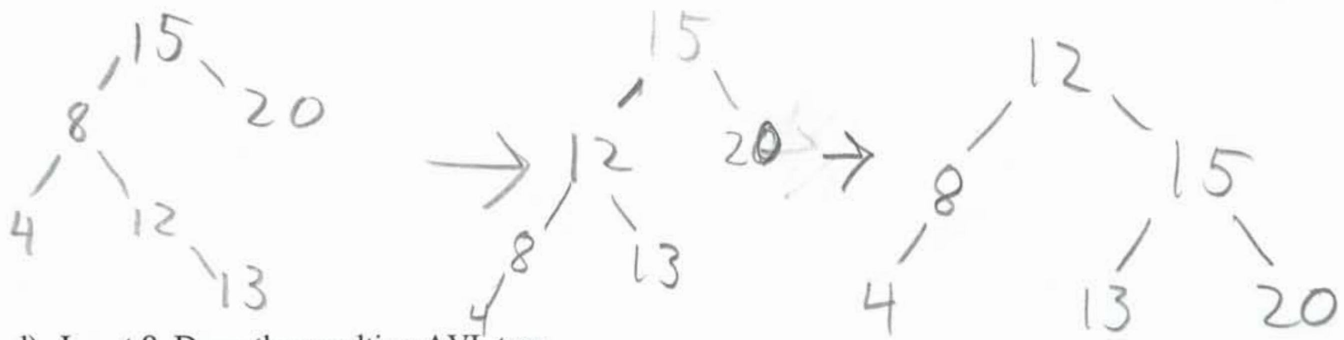
a) Insert the sequence 15, 12, 20, 8. Draw the final AVL tree.



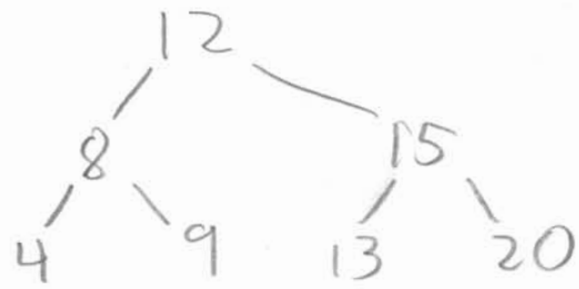
b) Insert 4. Draw the resulting AVL tree.



c) Insert 13. Draw the resulting AVL tree.



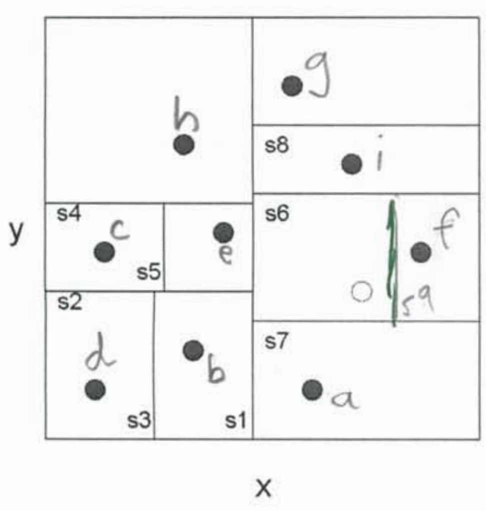
d) Insert 9. Draw the resulting AVL tree.

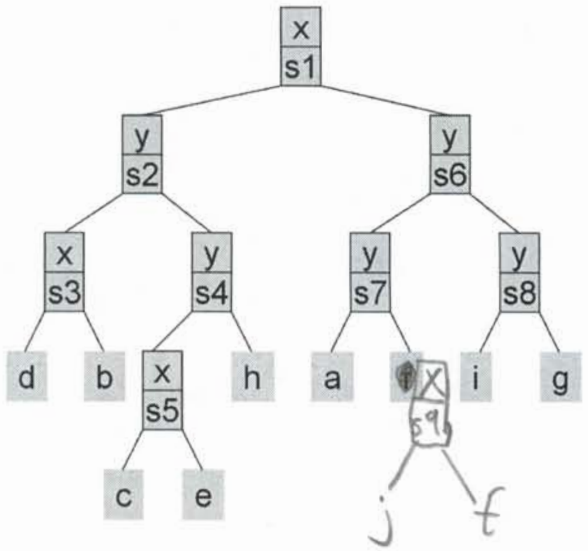


3

X)

Consider this k-d tree and the corresponding visualization of its elements.





a) The entries *a, b, c, d, e, f, g, h, i* are already labeled in the tree, but not in the visualization. They correspond to the solid circles.

Label each solid circle in the visualization according to the information provided in the tree. Write directly on the above visualization.

b) Draw what the tree would look like if the dashed circle were added as entry *j*.

Work with the given tree, do not build a new tree from scratch.

4
X)

Consider a set of objects with hash codes 40, 5, 18, 29, 35, 7 and a hash table of size 11.

a) Hash the above objects using separate chaining.

0	1	2	3	4	5	6	7	8	9	10
		35			5		40			

40
|
18
|
29
|
7

b) Hash the above objects using open addressing and linear probing.

0	1	2	3	4	5	6	7	8	9	10
		35			5		40	18	29	7

c) Hash the above objects using open addressing and quadratic probing.

0	1	2	3	4	5	6	7	8	9	10
29	7	35			5		40	18		

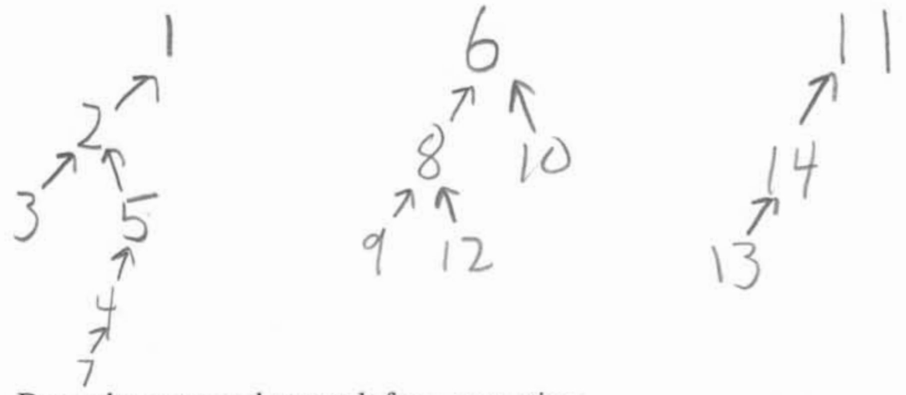
5

X)

The uptrees used to represent sets for manipulation by union-find algorithms can be stored in two *n*-element arrays: one giving the parent of each node (or -1 if the node has no parent), and the other giving the number of items in a set if the node is the root (representative node) of a set. For example, we can represent a collection of sets containing the numbers 1 through 14 as follows:

	1 ✓	2 ✓	3 ✓	4 ✓	5 ✓	6 ✓	7 ✓	8 ✓	9 ✓	10 ✓	11	12	13	14
up	-1	1	2	5	2	-1	4	6	8	6	-1	8	14	11
weight	6	–	–	–	–	5	–	–	–	–	3	–	–	–

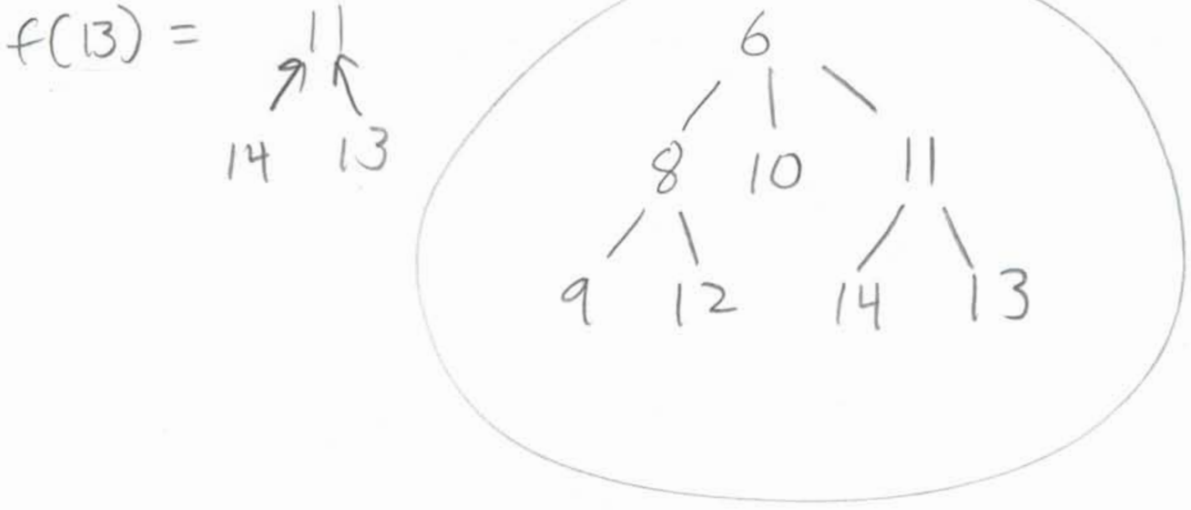
a) Draw a picture of the uptrees represented by the data in the above arrays.



b) Draw the uptrees that result from executing:

```
union(find(13), find(8));
```

Assume that the find operations use path compression and that the union operation uses union-by-size.

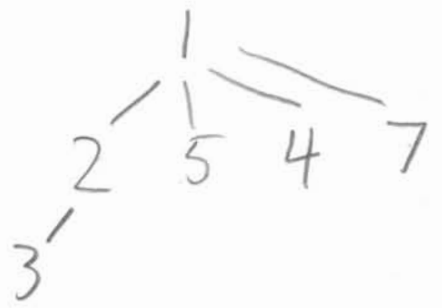


confusing?

c) Continuing based on your result for b), draw the uptrees that result from executing:

```
find(7);
```

Assume that the find operations use path compression.



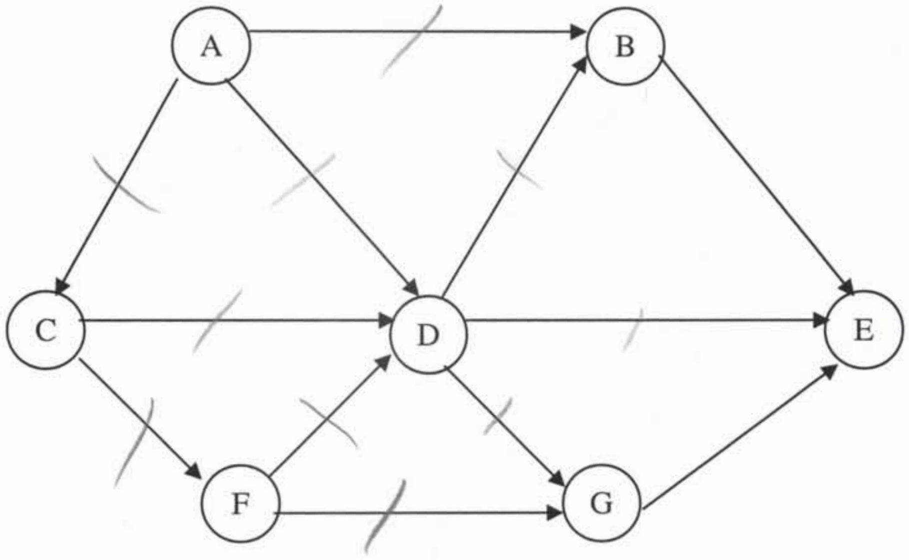
d) Show the contents of the two arrays after completing part *c)*.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
up	-1	1	2	1	1	-1	1	6	8	6	6	8	11	11
weight	6					8								

6 [illegible]

X)

Consider the following directed graph:



a) Perform a topological sort of the vertices.

$$\begin{tikzcd}
 & & & & B \arrow[r] & G \arrow[r] & E \\
A \arrow[r] & C \arrow[r] & F \arrow[r] & D \arrow[ur] \arrow[dr] & & & \\
 & & & & G \arrow[r] & B \arrow[r] & E
\end{tikzcd}$$

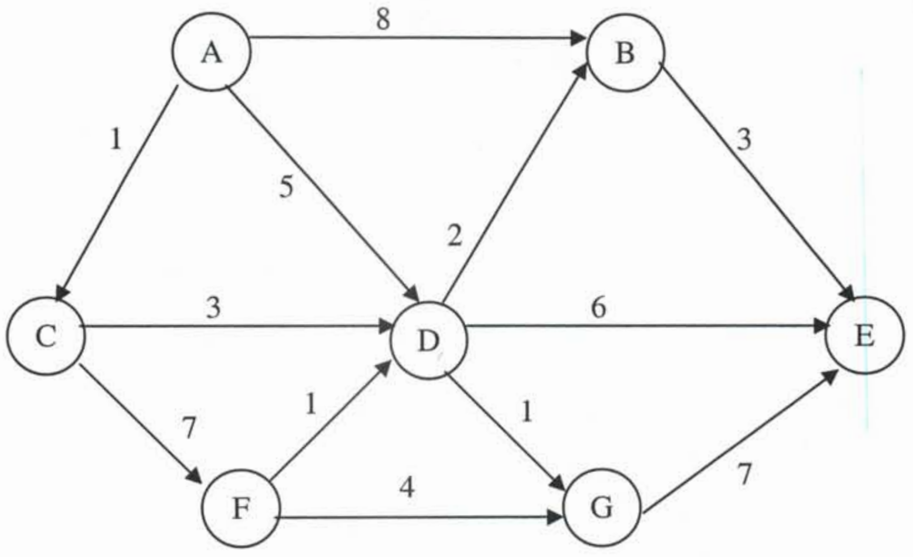
b) Is the sort you found unique? Why?

no, there is a point where multiple vertices have 0 in-degree.

7 ~~8~~

X)

Consider the following directed, weighted graph:



a) Step through Dijkstra's algorithm to calculate the single-source shortest paths from *A* to every other vertex. Show your steps in the table below. Cross out old values and write in new ones, from left to right within each cell, as the algorithm proceeds. Also list the vertices in the order which Dijkstra's algorithm marks them known:

Known vertices (in order marked known):

A C D G B F E

Vertex	Known	Distance	Path
A	✓	0 or N/A	N/A
B	✓	~~8~~ 6	~~A~~ D
C	✓	1	A
D	✓	~~5~~ 4	~~A~~ C
E	✓	~~10~~ 9	~~D~~ B
F	✓	8	C
G	✓	5	D

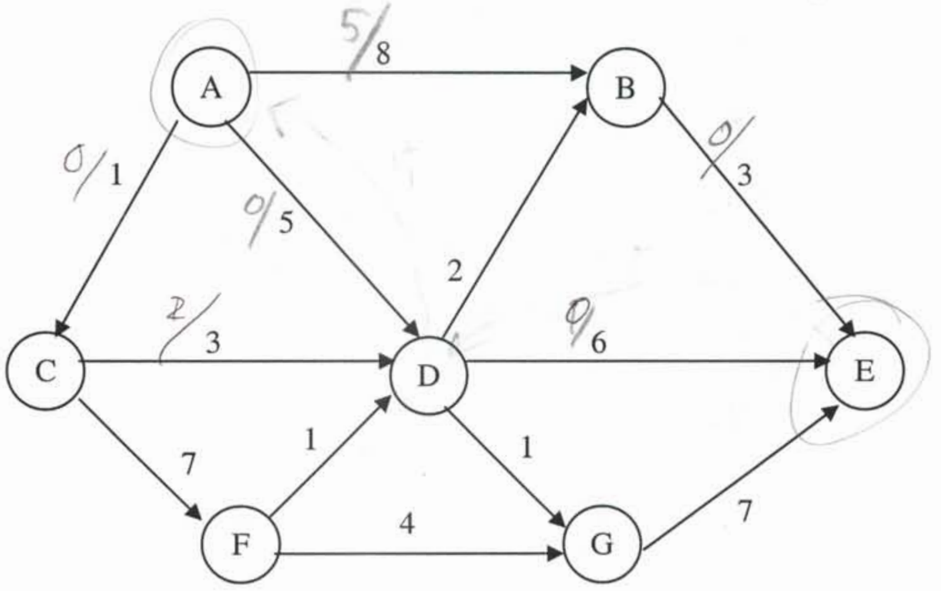
b) What is the lowest-cost path from *A* to *E* in the graph, as computed above?

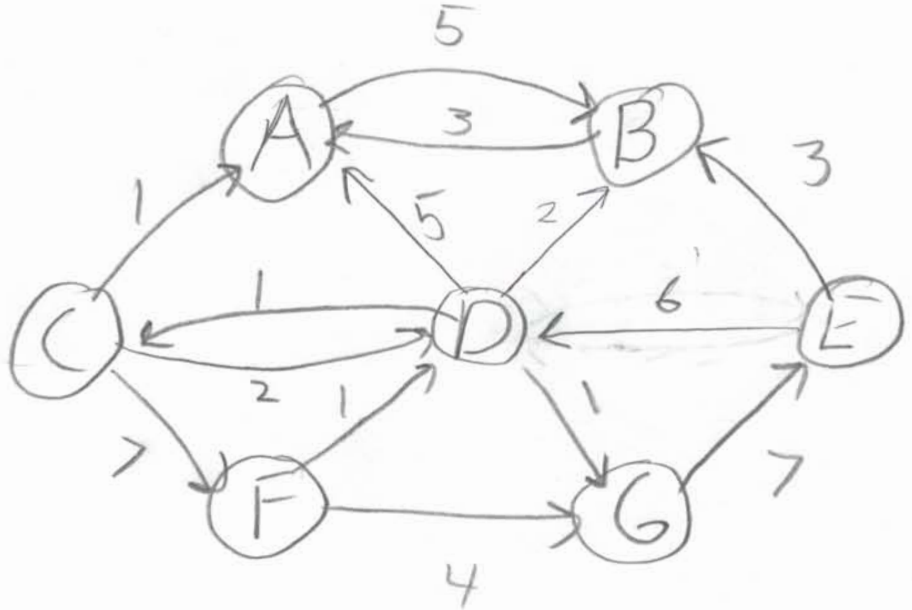
$A \rightarrow C \rightarrow D \rightarrow B \rightarrow E$ (cost = 9)

8 [illegible]

X)

Consider yet again the following directed, weighted graph:





ambiguous ... shortest 1st ?

You will need to compute a maximum flow from *A* to *E*, using the Ford-Fulkerson method. This page is for your work. Before running the algorithm, read ahead to know what information you need to provide for the different parts of this problem.

a) List each augmenting path you discover, in the order that you discover them, as well as the volume you are able to send down that path.

$A \rightarrow D \rightarrow E \quad (5)$

$A \rightarrow B \rightarrow E \quad (3)$

$A \rightarrow C \rightarrow D \rightarrow E \quad (1)$

↗ where?

b) Draw the residual graph that you have at the completion of the algorithm, when no more augmenting paths are available.

under original

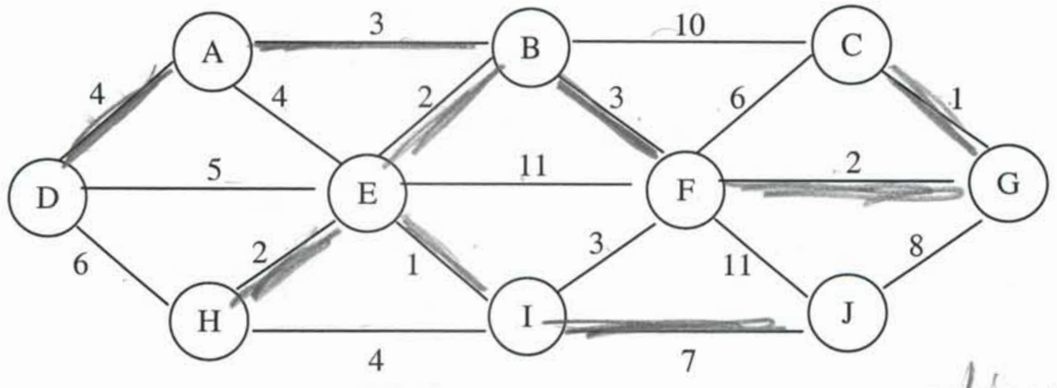
c) Give a minimum cut of the vertices in the graph, with source *A* and sink *E*.

$\{A, B\} \mid \{C, D, E, F, G\}$

9 ~~ ~~

X)

Consider the following undirected, weighted graph:



ambiguous order

Apply Kruskal's algorithm to compute a minimum spanning tree. In the designated spaces below, write down the edges in the order they are considered by Kruskal's algorithm. If the edge is part of the minimum spanning tree found by the algorithm, write it down in the first list of edges that form the MST. Write down the other edges considered by the algorithm in the order they were considered. Assume that the algorithm terminates as soon as the MST has been found.

In the lists, use (*x*,*y*) to indicate an edge connecting vertices *x* and *y*.

a) Edges that form part of the MST, in order considered:

(C,G) (E,I) (B,E) (E,H) (F,G)
(A,B) (B,F) (A,D) (I,J)

b) Other edges considered, but not included in the MST, in order considered:

(F,I) (A,E) (H,I) (D,E) (C,F)
(D,H) (G,J) (B,C) (E,F) (F,J)

c) Is the MST you found unique? Why?

No... I could have considered edges with the same weight in a different order. (F,I) could be in the MST, for example

~~50~~ 20 min to here.

10 **X)** 4

This problem is intended to be more difficult than most of this exam. You should probably be done earning the easier points available earlier in this exam before you begin on this question.

As part of developing a new anagram-based Web game that will make you a Web celebrity, you need to find all of the anagrams in the English language. Recall that an anagram is when the letters in one word can be rearranged to make another word. So the words "enlist", "inlets", "listen", and "silent" are all anagrams of each other.

You have a dictionary containing all words in the English language. So what you need is an efficient way to find which sets of words from that dictionary are anagrams. You ask Peter about an efficient way to do this, and he suggests using a hash table, but then must leave to go do whatever it is that graduate students do. If you wait until later to for Peter to have time to explain it, you may miss your chance on the Web, so you are on your own to figure it out.

maximum length? ~~length~~

a) Describe an algorithm that, given a dictionary containing *N* words, uses a hash table to find sets of anagrams in O(*N*) time and O(*N*) space. Pseudocode is not necessary, a simple description of the algorithm will suffice. See b) for a hint. overflow?

Store the hash value as computed in part (c) along with each string in the table. Use separate chaining. Then, when reading out the array, separate lists based on the stored hash values to obtain the sets.

b) Your algorithm requires the use of a hash function with particular properties. Describe the necessary properties of the hash function.

The hash function should depend on the number of each letter present only.

c) Give an appropriate example of a hash function with the necessary properties.

Have an array of the first 26 prime numbers. The hash function is the product of the prime numbers at the corresponding index.

11

X) [illegible]

This problem is intended to be the most difficult on this exam. You should probably be done earning the easier points available earlier in this exam before you begin on this question.

As you learned in Data Structures, the effective use of graph algorithms is often about figuring out how to set up your problem as an appropriate graph, so that you can then apply a standard algorithm. That knowledge has served you well, and your Web game startup company is doing awesome.

Your new problem is coming up with cute names for all those awesome games. You have hired a name person who sits around and comes up with cute names, and you have hired teams of designers that continue to come up with great games.

Because you have high standards for your company, you cannot deploy a game unless it has an appropriately cute name. And obviously the names are not so cute that people will find them interesting if there's no actual game. But while some names make sense for multiple games, not every name is appropriate for every game.

?

So, you have games $G_1 \ldots G_n$ and you have names $N_1 \ldots N_n$. You also have a list of names that are acceptable for each game. You must find the pairings of names with games that maximizes how many named games you can deploy.

For example:

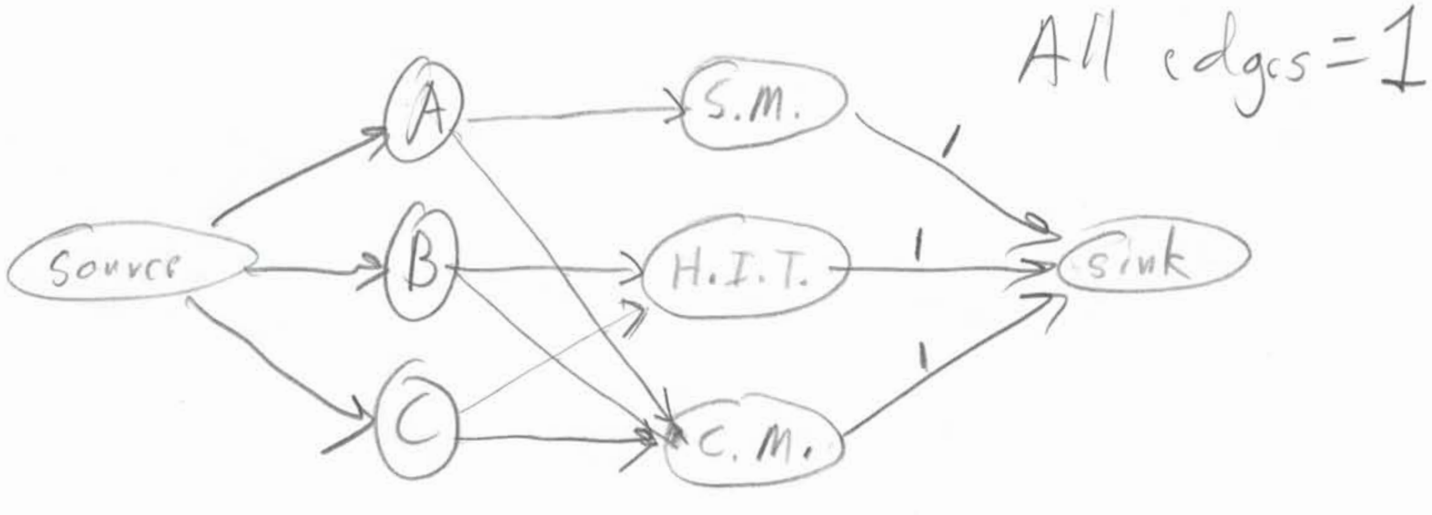
Game A might be named either "Soccer Mania" or "Crazy Monkeys"
(as it is a game about monkeys playing soccer).

Game B might be named either "Hang in There" or "Crazy Monkeys"
(as it is a game about monkeys hang gliding).

Game C might also be named either "Hang in There" or "Crazy Monkeys"
(as it is a game about monkeys hanging from the catwalks in the Paul Allen Center).

In this case, you should clearly name Game A "Soccer Mania". If you were to name Game A "Crazy Monkeys," you would only be able to deploy Game B or Game C, but not both. That would be bad for profits, and the world really does need more monkey-related Web games.

a) Draw a graph on which you could apply one of the standard graph algorithms discussed in class in order to solve this problem for the monkey-related examples.



b) Which algorithm do you apply to this graph in order to name the three games?

Max-Flow... middle edges selected represent namings.

c) Describe the general strategy for solving this problem given an arbitrary number of games, names, and potentially acceptable pairings. Pseudocode is not necessary, a simple description will suffice

A vertex G_i for each game and N_j for each name. An edge from game to name as given. Source to each game, each name to sink. All edges weigh 1.
Max-Flow, take all used edges from game vertex to name vertex as the naming.

$\frac{7}{10}$ total