Please use this Google doc during your interview (your interviewer will see what you write here). To free your hands for typing, we recommend using a headset or speakerphone.

PROBLEM:

A Binary Lock has N binary switches. The lock can be opened by flipping all switches to an "unlock" pattern. However, only some switch patterns are safe (you can access them in a global variable SAFE). Any other pattern will cause the lock to immediately lock. Additionally, switches can only be flipped one at a time.

Your task is to write a function that will accept a lock description: the current state of switches, a set of safe patterns, and the unlock sequence. Your function should return "unlock" if it's possible to unlock the Lock and "Cant open" otherwise.

Example 1:
  Current state: 010
  Unlock pattern: 111
  Safe patterns: 000 001 010 101 111

Correct response is: "Unlock" because there is a safe sequence to the open pattern:
010 -> 000 -> 001 -> 101 -> 111

Example 2:
 Current state: 00
 Unlock pattern: 11
 Safe patterns: 00 11

Correct response is: "Cant open"

NOTES:

Unlock a lock

Trie, BFS
Graph

INPUT:

The current state
Set of safe patterns -> s
Unlock sequence

OUTPUT:

Unlock and Cant open

EXAMPLE:

Example 1:
  Current state: 010
  Unlock pattern: 111
  Safe patterns: 000 001 010 101 111

Correct response is: "Unlock" because there is a safe sequence to the open pattern:
010 -> 000 -> 001 -> 101 -> 111

RUNNIG

Safe patterns: 000 001 010 101 111

State -> Keys
010 -> 110,000,011

000-> 100,010,001

010-> 110, 000, 011

001 -> 101,011,000

101 -> 001,111,100

APPROACH:

1 -> BFS Queue - Time: O(s) Space: O(s-checked s)
Generate the keys for the initial state.
Generate a BFS Queue with initial states.
While the BFS Queue is not empty, poll first state, if it is not in the safe set continue to the next state, if it is on the safe set generate the newKey from the actual state, if we reach the unlock pattern return unlock if not add the state to the BFS queue.
Return cant open.

CODE:

```java
/**
 * @autor Oscar Camacho
 * Google Java Style
 * Approach - 1 -> BFS Queue - Time: O(new states*s.length()+(s)) Space:
O(s-checked s)
 */

 // TODO
 // package com.example;
 // import example;

class Solution {
  public String unlockSequence(String currentState, String unlockPattern,
      Set<String> safePatterns) {
    Queue<String> newKeys = generateNewKeys(currentstate);

    for (String key : newKeys) {
      if (key.equals(unlockPatterns)) {
        return "Unlock";
      }
    }

    Queue<String> keysToExplore =  new LinkedList(newKeys);

    while (!keysToExplore.isEmpty()) {
      currenState = keysToExplore.poll();
      newKeys = generateNewKeys(currentState);

for (String key : newKeys) {
  if (!faePatterns.contains(key)) {
    continue;
 }

 safePatterns.remove(key)
        if (key.equals(unlockPatterns)) {
          return "Unlock";
        }
        keysToExplore.add(key);
      }

    }

    return "Cant open";
  }

  public Queue<String> generateNewKeys(String state)  { // Time: O(state.length())
```

```
    Queue<String> newKeys = new LinkedList();

    for (int i = 0 ; i < state.length() ; i++) {
      char at = state.charAt(i);
      if (at == '0') {
        at = '1';
      } else {
        at = '0';
      }

      newKey.add(s.substring(0, i) + at + s.substring(i+1));
    }

    return newKeys;
  }
}
```

TEST:

OPTIMIZATION:

Please use this Google doc during your interview (your interviewer will see what you write here). To free your hands for typing, we recommend using a headset or speakerphone.

PROBLEM:

1.) Let's create a redacted version of an English-language document. The original document is stored as a plain-text file. Certain words are prohibited and must be redacted; these words are given in another plain-text file with one word per line.

Please implement a program that reads both files and writes a redacted document as output.

NOTES:

Scanner
BufferReader

INPUT:

String originalFilePath -> of
String wordsFilePath -> wl
  50,000

OUTPUT:

EXAMPLE:

```
original
one two three four five

word
two five

output
one-three-four

output
one-***-three-four-***
```

APPROACH:

1 -> Load full wordList and check word by word original file - Time: O(wl+of) Space: O(wl+1 word)
Load the wordList.
Read word by word from the original file, if the word is in the wordList write "" if not write the word, check spaces.***

CODE:

```java
/**
 * @autor Oscar Camacho
 * Google Java Style
 * Approach 1 -> Load full wordList and check word by word original file - Time: O(wl+of) Space: O(wl+1 word)
 */

 // TODO
 // package com.example;
 // import example;

class Solution {
  public void redactWords(String originalFilePath, String wordListFilePath) {
    // Load wordList
    Scanner sc = new Scanner(wordListFilePath);
    Set<String> wordList = new HashSet();
    while (sc.hasNext()) {        // May be a bug
      String word = sc.next();
```

```
      wordList.add(word);
    }

    // Check word by word from the original file
    sc = new Scanner (originalFilePath);
    BufferWriter bf = new BufferWriter("NewFile.txt");

    String word = sc.next();
    if (wordList.contains(word)) {
      bf.add("***");
    } else {
      bf.add(word);
    }

    while (sc.hasNext()) {
      word = sc.next();
      if (wordList.contains(word)) {
        bf.add(" ***");
      } else {
        bf.add(" "+word); // Add to the buffer
        bf.push() // Write in the file
      }
    }

    sc.close();
    bf.close();
  }
}
```

TEST:

```
inputFile.txt
couldHave/wordList.txt
out -> NewFile.txt, fixed file in the function

main() {
  redactWords(inputFile.txt, couldHave/wordList.txt) {

  }
}
```

OPTIMIZATION:

As for a file path to write

There are a number of piles of coins. Each coin has a different value. Each time the user is allowed to pick the top coins from one of any pile. Please write an algorithm to return the maximum value the user can get by picking n times.

5 1 7
2 10 3
4 2 1

max = 11.

n = 2

NOTES:

INPUT:

`List of stacks`

OUTPUT:

`int max`

EXAMPLE:

5 1 7
2 10 3
4 2 1

n= 2
7,5
max = 12.

n=3
7, 5, 3 -> Error
max = 18

n = 2
1 5   1   7
2 7   11   10

n = 3
1 7   5   1
2 11   10   7
3 13   13   11 -> Error can not sort here

APPROACH:

1 -> Explore all possibilities, recursion and backtracking - Time: O(2^n) Space: O(2^n) (stack)
Recurse, the every point explore taking the coin and exploring continue without the coin, update the maxValue

Explanation
Set a maxValue to 0
Recurse, while I still can have space for another coin, explore thanking a conin and explore not tanking it.

2 -> Heap approach
Build a List of heap and explore tacking coins

CODE:

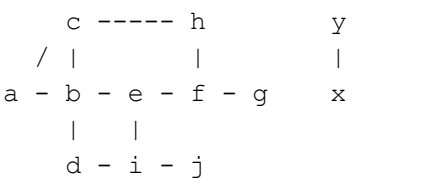
```
/**
 * @autor Oscar Camacho
 * Google Java Style
 * Approach 1 -> Explore all possibilities, recursion and
backtracking - Time: O(2^n) Space: O(2^n) (stack)
 */

 // TODO
 // package com.example;
 // import example;
```

```
class Solution {
  static int maxValue;
  static List<Deque<Integer>> coinsStacks;
  static int coinsLimit;
  public int maxValue(List<Deque<Integer>> coinsStacks, int
        coinsLimit) {

    this.coinsStacks = coinsStacks;
    this.coinsLimit = coinsLimit;
    maxValue = 0;

    recurse(0, 0, 0);

    return maxValue;
  }

  public void recurse(int indexStack, int coinsCount, int sumValue){
    if (indexStack => coinsStack.size()){
      return;
    }

    if (coinsCount == coinsLimit) {
      maxValue = Math.max(maxValue, sumValue);
      return;
    }

    while (!coinsStack.get(indexStack).isEmpty()) {
      int atValue = coinsStack.get(indexStack).pop();
      // Exploring taking the coin;
      recurse (indexStack, coinsCount + 1, sumValue + atValue);
      // Exploring no taking the coin;
      coinsStack.get(indexStack).push();
      recurse (indexStack + 1, coinsCount, sumValue);
    }
    return;
  }
}
```

TEST:

OPTIMIZATION:

Please use this Google doc during your interview (your interviewer will see what you write here). To free your hands for typing, we recommend using a headset or speakerphone.




```
Friends(b) -> a, c, d, e
Mutual_friends(b, i) -> d, e
Mutual_friends(b, f) -> e
```

NOTES:

INPUT:

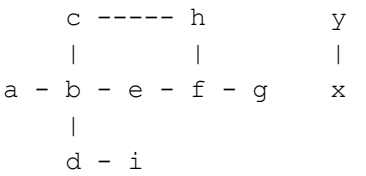
Graph Person[List<Friend>] -> g
Person

OUTPUT:

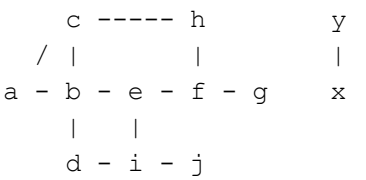
Friends in common
List of people
Most mutual friend

Given a social network and an individual A in the social network, return a list of individuals that have the most mutual friends with individual A.

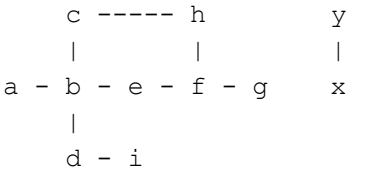
EXAMPLE:




```
Ind = b

b -> a,c,d,e

Person - Friends

b -> a,c,d,e

i -> d -> 1
e -> b,f -> 0
f -> e,h,g -> 1

ans = d

{ 'i': 1, 'f': 1, 'h': 1 }
```




```
b -> a,c,d,e

i -> d -> 1
e -> b,f -> 0
f -> e,h,g -> 1
```




```
b -> a,c,d,e
```

```
a -> b
c -> b, h
d -> b, i
e -> b, f
```

APPROACH:

1 -> Intersect input person with all other persons - Time: O(g) Space: O(n)
For each person different than the input person compare all their friends with the input person and save the intersection

2 -> Check Friend - Time: O(g^2) Space: O(g)
Check all input person's friends, all the friends of input person different than the input person will be a mutual friend

Traversal
Set
retainAll, cotains

CODE:

```
Set<Integer> mutualFriend(ArrayList<List<Integer>> graph, int person) {
  Set<Integer> ans = new HashSet();
  List<Integer> inputPersonFriends = graph.get(person);

  for (int friend : inputPersonFriends) {
    for (int mutualFriend : grap.get(friend)) {
      ans.add(mutualFriend);
    }
  }

  ans.remove(person;)
  return ans;
}
```

```
    d
  / | \
a - b - c
    |
    e

a - b - c - d
```

TEST:

OPTIMIZATION: