Interview Notes - Dropbox

#15-Mar-2018 #Internship #Onsite

Behavioural

https://www.paysa.com/blog/2017/04/05/what-to-expect-from-the-dropbox-interview-process/

Some examples of things that greatly help us understand your technical abilities are

- Github projects
- Open source contributions
- Apps, websites, or other side projects that you built for fun
- Papers that you have published
- Products that you have worked on in the past
- Links to your blog or any other technical writing

Our interviewers will try to make time in every interview to chat about your achievements so that we can understand your unique strengths as an engineer. We hope you'll be excited and ready to tell us about your past work.

Engineering Values

1. Relentlessly focus on impact. Iterate, prioritize, and build products that solve real user needs.
2. Own it. Do whatever it takes to see a project through to completion. Deliver on promises and own up to failure.
3. Pursue simplicity and sweat the details. Design and develop products anyone can use.
4. Build a culture you want to be part of ten years from now. Collaborate and help your team. Seek out feedback and provide constructive feedback to others.

> "Our hiring philosophy centers on our company values and candidates who can speak to them often do well. For example, we focus on teamwork and trust. Building a product useful to millions of people is truly a team effort — whether you're an engineer, designer, marketer, or anything else. Our users rely on Dropbox to help simplify their lives, and we take that trust seriously."

Questions:

1. Hobbies other than work and study?
2. Some questions about resume
3. Why do you want to work in dropbox?
   It's the culture, the **creative energy**, and the opportunity to tackle **amazing challenges** that makes working at Dropbox so awesome.
   > *What I like about this company is that our employees really take the time to listen to a problem before trying to offer a solution. -Genevieve, Product Manager*

> *The work we do every day is so mission critical to our customers' satisfaction. It's pretty exciting to be on for the ride.   -Ruchi, Engineering Manager*

- People, everyone is positive, very human place
- Culture is daunting and magical: lots of dedication and hard-work with fun; deliver on results, never take themselves seriously; respect my time at home, seamless life and work. always surprised;
- Believe users own their data; what you built has so much users impact;
- Perks: gym; ...
- Not a place come to work but come to grow.

https://youtu.be/-ZuxQcp84o0

Questions to Ask

1. Could you share some exciting amazing tech success in Dropbox? For example, how did dropbox built its own infrastructure (move away from AWS) with a small team?
2. People share that they are always surprised working at Dropbox. Could you share some of the fun or exciting moments?
3. Not a place to work but a place to grow, what are some of the training opportunities out there or how do people grow their technical skills?

4. The company is adding more excitement to it e.g. Dropbox Paper, what other features will be coming? what's the strategic direction that the company is heading to?
5. Collaborate & help your teammates, as a team, or how do cross functional collaborates happen, any company-wide initiatives?
6. What are some of the important skills that you learnt from working at Dropbox? Or, what are some of the skills to succeed in working environment in Dropbox?
7. What are the tech stacks for the application dropbox offered? For example, the desktop application.
8. How's dropbox's team structure? I mean what are some of the teams and what are they responsible for? group by products etc?
9. What are the development lifecycle / methodology for the produce team e.g. how quick do they prototype or iterate quick to incorporate market changes/ customers needs.
10. How does your team train up new recruits?
11.

Algorithm

Allocate ID / Phone Directory

Things to consider:

- what happens if all IDs used up? throws Exception or return -1
- release invalid ID or release id that is not allocated, how to handle? just return or throw Exception??

FreeList approach:

- Space is O(n), heavy, because of the data structure of queue and set too
- O(1) time in allocate and release

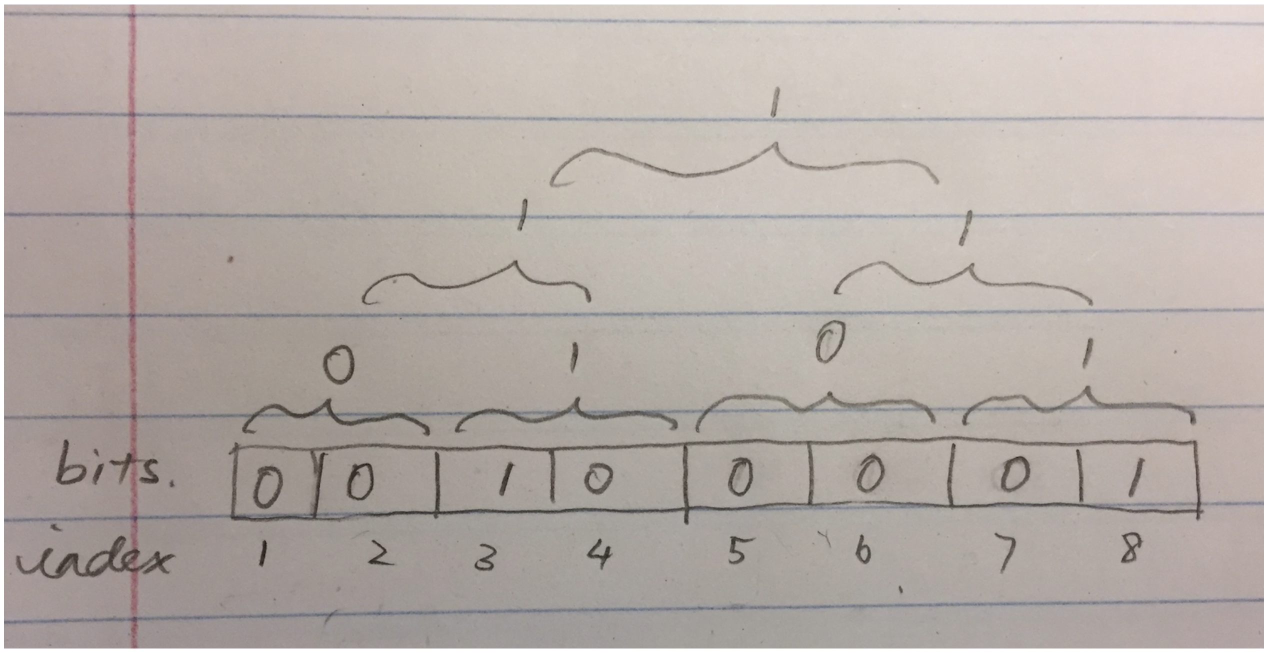
```java
public class Allocator{
    private Queue<Integer> freeList;
    private Set<Integer> allocated;
    private final int MAX_ID;

    public Allocator(int maxId) {
        this.MAX_ID = maxId;
        this.freeList = new LinkedList<>();
        for(int i=0; i<maxId; i++) {
            freeList.offer(i);
        }
        this.allocated = new HashSet<>();
    }

    public int allocate() {
        if(freeList.isEmpty())
            return -1;
        int id = freeList.poll();
        allocated.add(id);
        return id;
    }

    public void release(int id) {
        if(id<0 || id>=MAX_ID || !allocated.contains(id)) return;
        allocated.remove(id);
        freeList.add(id);
    }

    public boolean check(int id) {
        if(id<0 || id>=MAX_ID) return false;
        return !allocated.contains(id);
    }

    public static void main(String[] args) {
        Allocator allocator = new Allocator(10);
        int id1 = allocator.allocate();
        int id2 = allocator.allocate();
        int id3 = allocator.allocate();
        System.out.println(id1+", "+id2+", "+id3);

        System.out.println(allocator.check(id1));
        System.out.println(allocator.check(id2));
        System.out.println(allocator.check(id3));
        System.out.println(allocator.check(11));
        System.out.println(allocator.check(-1));

        allocator.release(2);
        System.out.println(allocator.check(id3));

    }
}
```

*If no need to maintain a ordering allocation, can use just one set called available to achieve the above. Allocate is get and remove the first element in the set. (available.iterator().first())

BitSet approach:
Space is much efficient: O(c), wouldn't say it is O(1) because we still need max_size number of bits
Time: **worst case O(n) in searching for next available id**

```java
public class Allocator{
    private final int MAX_ID;
    private BitSet bitSet;
    private int nextAvailable;

    public Allocator(int maxId) {
        this.MAX_ID = maxId;
        this.bitSet = new BitSet(maxId);
        this.nextAvailable = 0;
    }

    public int allocate() {
       if(nextAvailable==MAX_ID) return -1;
       int num = nextAvailable;
       bitSet.set(num);
       nextAvailable = bitSet.nextClearBit(num);
       return num;
    }

    public void release(int id) {
        if(id<0 || id>=MAX_ID) return;
        if(bitSet.get(id)) {
            bitSet.clear(id);
            nextAvailable = Math.min(nextAvailable, id);
        }
    }

    public boolean check(int id) {
        if(id<0 || id>=MAX_ID) return false;
        return !bitSet.get(id);
    }
}
```

Using the idea of binary tree (in heap array representation in bitset):
Space: 2 * space used in bitset
Time: O(log n) time in **allocate and release**



```
public class Allocator{
    private final int MAX_ID;
    private BitSet bitSet;

    public Allocator(int maxId) {
        this.MAX_ID = maxId;
        this.bitSet = new BitSet(maxId*2-1);
    }

    public int allocate() {
        int index=0;
        while(index<MAX_ID-1) {
            if(!bitSet.get(index*2+1)) {
                index = index*2+1;
            } else if(!bitSet.get(index*2+2)) {
                index = index*2+2;
            } else {
                return -1;
            }
        }

        bitSet.set(index);
        updateTree(index);

        return index-MAX_ID+1;
    }

    public void updateTree(int index) {
        while(index>0) {
            int parent = (index-1)/2;
            if(index%2==1) { //left child
                if(bitSet.get(index) && bitSet.get(index+1)) {
                    bitSet.set(parent);
                } else {
                    bitSet.clear(parent); //it is required for release id
                }
            } else {
                if(bitSet.get(index) && bitSet.get(index-1)) {
                    bitSet.set(parent);
                } else {
                    bitSet.clear(parent);
                }
            }
            index = parent;
        }
    }

    public void release(int id) {
        if(id<0 || id>=MAX_ID) return;
```

```
            if(bitSet.get(id+MAX_ID-1)) {
                bitSet.clear(id+MAX_ID-1);
                updateTree(id+MAX_ID-1);
            }
        }

        public boolean check(int id) {
            if(id<0 || id>=MAX_ID) return false;
            return !bitSet.get(id+MAX_ID-1);
        }
    }
```

Download File - BitTorent

Things to consider:

- 增加一个显示下载百分比的功能 (use BitSet can do this)

If all chunks are given:

- just do something like interval merge

```
public boolean isFileDone(List<Chunk> chunks, int size) {
    if(chunks==null || chunks.size()==0) return false;

    Collections.sort(chunks, (a, b)-> a.start - b.start);
    int start = chunks.get(0).start, end = chunks.get(0).end;
    for(int i=1; i<chunks.size(); i++) {
        Chunk chunk = chunks.get(i);
        if(chunk.start>end)
            return false;
        else
            end = Math.max(end, chunk.end);
    }

    return start==0 && end==size;
}
```

If chunks are coming in stream:
Using Priority Queue, addBlock is O(klog(m)) depending on the chunk arrivals and merging, but amortised worst case should be no more than O(logn).

```java
class Downloader {
    private PriorityQueue<Chunk> chunks;
    int size;

    public Downloader(int size) {
        this.size = size;
        chunks = new PriorityQueue<>((a,b)->a.start-b.start);
    }

    public void addBlock(Chunk chunk) {
        chunks.offer(chunk);
        if(chunks.size()>1) {
            Chunk smallest = chunks.poll();
            while(!chunks.isEmpty() && chunks.peek().start <= smallest.end
) {
                Chunk c = chunks.poll();
                smallest.end = Math.max(smallest.end, c.end);
            }
            chunks.offer(smallest);
        }
    }

    public boolean isDone() {
        if(chunks.isEmpty()) return false;
        if(chunks.peek().start==0 && chunks.peek().end==size) return true;
        return false;
    }
}
```

Using BitSet

```java
class DownloaderBitSet {
    private BitSet chunksBitSet;
    int size;

    public DownloaderBitSet(int size) {
        this.size = size;
        chunksBitSet = new BitSet(size);
    }

    public void addBlock(Chunk chunk) {
        chunksBitSet.set(chunk.start, chunk.end);
    }

    public boolean isDone() {
        return chunksBitSet.cardinality()==size;
    }
}
```

TreeSet approach
https://www.programcreek.com/2014/08/leetcode-data-stream-as-disjoint-intervals-java/

```
public class SummaryRanges {

    TreeSet<Interval> set;

    /** Initialize your data structure here. */
    public SummaryRanges() {
        set = new TreeSet<Interval>(new Comparator<Interval>(){
            public int compare(Interval a, Interval b){
                return a.start-b.start;
            }
        });
    }

    public void addNum(int val) {
        Interval t = new Interval(val, val);

        Interval floor = set.floor(t);
        if(floor!=null){
            if(val<=floor.end){
                return;
            }else if(val == floor.end+1){
                t.start = floor.start;
                set.remove(floor);
            }
        }

        Interval ceil = set.higher(t);
        if(ceil!=null){
            if(ceil.start==val+1){
                t.end = ceil.end;
                set.remove(ceil);
            }
        }

        set.add(t);
    }


    public List<Interval> getIntervals() {
        return Arrays.asList(set.toArray(new Interval[0]));
    }
}
```

BST/Interval Tree??? how?

Duplicate Files

Things to consider:

1. Symbolic link, same file/dir with diff name, cannot detect cycle by visited...cycle? **use absolute path/ skip symbolic link (if we search the whole file system)**
2. invalid or malformed files e.g. permission or cannot read
3. compare file by hashing, MD5 false positive (due to hash collision), use SHA256/512 very very little chance of collision

4. ~~If dir depth is large: DFS might stack overflow, use BFS; the variable to store pathname might overflow.~~
5. Can ask question after interview how Dropbox solve this type of problem.
6. Most memory consuming: MD5, read in files etc
7. Race conditions?: someone is writing the file while you are reading etc
8. If error / hanging up in between: checkpoints, save states from time to time

```java
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.*;

public class FindDuplicateFiles {

    public List<List<String>> findDuplicates(String path) throws Exception {
        List<List<String>> result = new ArrayList<>();
        if(path==null || path.length()==0) return result;

        List<String> filesPath = getAllFiles(path);
        Map<String, List<String>> dupFilesMap = new HashMap<>();

        for(String filePath: filesPath) {
            File file = new File(filePath);
            String hashCode = hashFile(file, "MD5");

            if(!dupFilesMap.containsKey(hashCode)) {
                dupFilesMap.put(hashCode, new ArrayList<>());
            }
            dupFilesMap.get(hashCode).add(filePath);
        }

        for(List<String> dupFiles: dupFilesMap.values()) {
            if(dupFiles.size()>1)
                result.add(dupFiles);
        }

        return result;
    }

    private List<String> getAllFiles(String path) {
        List<String> filesPath = new ArrayList<>();
        Stack<String> s = new Stack<>();
        s.push(path); //DFS with Stack

        while(!s.isEmpty()) {
            String currPath = s.pop();
            File file = new File(currPath);

            if(file.isFile()) {
                filesPath.add(currPath);
            } else if(file.isDirectory()) {
                String[] subDir = file.list();
                for(String dir:subDir) {
                    s.push(currPath+"/"+dir);
                }
            }
        }

        return filesPath;
    }

    public List<List<String>> findDuplicatesOpt(String path) throws Except
ion{
        List<List<String>> result = new ArrayList<>();
        if(path==null || path.length()==0) return result;

        Map<Long, List<String>> fileSizeMap = getAllFilesBySize(path);
        Map<String, List<String>> dupFilesMap = new HashMap<>();

        for(List<String> filePaths: fileSizeMap.values()) {
            if(filePaths.size()<2) continue;
            for(String filePath: filePaths) {
                File file = new File(filePath);
                String hashCode = hashFile(file, "MD5");

                if (!dupFilesMap.containsKey(hashCode)) {
                    dupFilesMap.put(hashCode, new ArrayList<>());
                }
                dupFilesMap.get(hashCode).add(filePath);
            }
        }

        for(List<String> dupFiles: dupFilesMap.values()) {
            if(dupFiles.size()>1)
                result.add(dupFiles);
        }

        return result;
    }

    private Map<Long, List<String>> getAllFilesBySize(String path) {
        Map<Long, List<String>> fileSizeMap = new HashMap<>();
        Stack<String> s = new Stack<>();
        s.push(path);

        while(!s.isEmpty()) { //DFS by stack
            String currPath = s.pop();
            File file = new File(currPath);

            if(file.isFile()) {
                long size = file.length();
                if(!fileSizeMap.containsKey(size))
```

```
                    fileSizeMap.put(size, new ArrayList<>());
                fileSizeMap.get(size).add(currPath);
            } else if(file.isDirectory()) {
                String[] subDir = file.list();
                for(String dir:subDir) {
                    s.push(currPath+"/"+dir);
                }
            }
        }

        return fileSizeMap;
    }

    private static String hashFile(File file, String algorithm)
            throws Exception {
        try (FileInputStream inputStream = new FileInputStream(file)) {
            MessageDigest digest = MessageDigest.getInstance(algorithm);

            byte[] bytesBuffer = new byte[1024];
            int bytesRead = -1;

            while ((bytesRead = inputStream.read(bytesBuffer)) != -1) {
                digest.update(bytesBuffer, 0, bytesRead);
            }

            byte[] hashedBytes = digest.digest();

            return convertByteArrayToHexString(hashedBytes);
        } catch (NoSuchAlgorithmException | IOException ex) {
            throw new Exception(
                    "Could not generate hash from file", ex);
        }
    }
    private static String convertByteArrayToHexString(byte[] arrayBytes) {
        StringBuffer stringBuffer = new StringBuffer();
        for (int i = 0; i < arrayBytes.length; i++) {
            stringBuffer.append(Integer.toString((arrayBytes[i] & 0xff) + 0x100, 16)
                    .substring(1));
        }
        return stringBuffer.toString();
    }

    public static void main(String[] args) throws Exception{
        List<List<String>> dupFiles = new FindDuplicateFiles().findDuplicatesOpt("./Resources/Dropbox");
        for(List<String> dup: dupFiles) {
            System.out.println(dup.toString());
        }

        //new FindDuplicateFiles().read1KBEachTime("./Resources/Dropbox/board.txt");
    }


}
```

How to check the file types in Linux:
Linux has different APIs e.g. is_regular_file(), is_block_file etc

How to hash 1KB each time:

```
  public void read1KBEachTime(String path)
            throws FileNotFoundException, IOException{
        File file = new File(path);
        FileInputStream fis = new FileInputStream(file);
        byte[] buffer = new byte[1024];
        int count;
        while((count=fis.read(buffer))!=-1) {
           // hash(buffer);
            System.out.print(buffer);
            count = fis.read(buffer);
        }
    }
```

```
    - • //Helper functions (need to be defined by you)
    - public List<String> getFiles(String directoryPath); //Returns all files directly under this directory
    - public List<String> getDirectories(String directoryPath); //Returns all directories directly under this directory
    - public String getFileContent(String filePath); //Returns content of a file
    - //Write this function (Interviewer just cared about this)
    - public List<String> findDuplicates(String rootDirectory) {
    -  ...
    -  ...
```

609. Find Duplicate File in System

Example 1:

```
Input:
["root/a 1.txt(abcd) 2.txt(efgh)", "root/c 3.txt(abcd)", "root/c/d 4.txt(efgh)", "root 4.txt(efgh)"]
Output:
[["root/a/2.txt","root/c/d/4.txt","root/4.txt"],["root/a/1.txt","root/c/3.txt"]]
```

```
class Solution {
    public List<List<String>> findDuplicate(String[] paths) {
        Map<String, List<String>> dupFilesMap = new HashMap<>();

        for(String dir: paths) {
            String[] fileAndContent = dir.split(" ");
            String dirPath = fileAndContent[0];
            for(int i=1; i<fileAndContent.length; i++) {

                String fileName = fileAndContent[i].substring(0, fileAndContent[i].indexOf("("));
                String fileContent = fileAndContent[i].substring(fileAndContent[i].indexOf("(")+1, fileAndContent[i].indexOf(")"));
                if(!dupFilesMap.containsKey(fileContent))
                    dupFilesMap.put(fileContent, new ArrayList<>());
                dupFilesMap.get(fileContent).add(dirPath+"/"+fileName);
            }
        }

        List<List<String>> result = new ArrayList<>();
        for(List<String> dupFiles: dupFilesMap.values()) {
            if(dupFiles.size()>1)
                result.add(dupFiles);
        }

        return result;
    }
}
```

Find Byte in File

Simple substring match

```
public boolean containBytes(byte[] pattern, byte[] text) {
    for(int i=0; i<=text.length-pattern.length; i++) {
        int j=0;
        while(j<pattern.length && pattern[j]==text[i+j]) {
            j++;
        }
        if(j==pattern.length) return true;
    }

    return false;
}
```

Use rolling hash to check substring match

- mod a large prime number to avoid overflow
- the parameters a and prime number determines the collision rate. We want the slot large enough, N2 *such that the expected collision for n elements is just 1/N, there are n numbers, so total expected collision is n*1/n =1

- We need to check if this is really the substring match if the hash code is the same

```java
public boolean containsBytesRollingHash(byte[] pattern, byte[] text) {
    if(text.length<pattern.length)
        return false;

    int m = pattern.length, n=text.length;
    byte[] initialBytes = Arrays.copyOfRange(text, 0, m);
    RollingHash hashFun = new RollingHash(31, initialBytes);

    long patternHashVal = hashFun.hash(pattern);
    for(int i=0; i<=n-m; i++) {
        if(patternHashVal==hashFun.currHashValue) {
            //need to check byte by byte to ensure
            int j=0;
            while(j<m && pattern[j]==text[i+j]) j++;
            if(j==m) return true;
        }

        if(i<n-m){
            hashFun.recompute(text[i], text[i+m]);
        }
    }

    return false;
}

public boolean containsBytesFileRollingHash (byte[] pattern, File file) throws FileNotFoundException, IOException {
    FileInputStream fis = new FileInputStream(file);

    try {
        byte[] initialBytes = new byte[pattern.length];

        if (fis.read(initialBytes) != pattern.length) return false;

```

```
            RollingHash hashFun = new RollingHash(31, initialBytes);
            long patternHashVal = hashFun.hash(pattern);
            if (patternHashVal == hashFun.currHashValue) return true;

            Queue<Byte> window = new LinkedList<>();
            for (int i = 0; i < initialBytes.length; i++) {
                window.offer(initialBytes[i]);
            }

            int b;
            while ((b = fis.read()) != -1) {
                System.out.println(b);

                hashFun.recompute(window.poll(), (byte) b);
                window.offer((byte) b);
                if (patternHashVal == hashFun.currHashValue) return true;
            }

        }finally {
            fis.close();
        }

        return false;
}

class RollingHash {
    private final int LARGE_PRIME = 105613;
    private final int a;
    private int h=1;
    private final int WINDOW_LENGTH;

    private long currHashValue;

    public RollingHash(int a, byte[] initialBytes) {
        this.a = a;
        this.WINDOW_LENGTH = initialBytes.length;

        // The value of h would be "pow(a, WINDOW_LENGTH-1)%q
        for (int i = 0; i < WINDOW_LENGTH-1; i++) {
            //a^n % p = (a^n-1 % p * a%p)%p;
            h = (h*a)%LARGE_PRIME;
        }

        currHashValue = hash(initialBytes);

    }

    public long hash(byte[] bytes) {
        int hashVal=0;

        for(int i=0; i<bytes.length; i++) {
            hashVal = (a*hashVal + bytes[i])% LARGE_PRIME;
        }

        return hashVal;
    }

    public long recompute(byte removed, byte incoming) {

        //(a+b) %p = (a%p + b%p) %p
        //(a-b) %p = (a%p - b%p) %p might give negative
        //(a*b) %p = (a%p * b%p) %p
        currHashValue = (a*(currHashValue - removed*h) + incoming)%LARGE_PRIME;

        // We might get negative value of t, converting it to positive
        if (currHashValue < 0)
            currHashValue += LARGE_PRIME;

        return currHashValue;
    }
}
```

Top K Photo ID

- counting the freq is O(n)
- use selection algorithm to select top k element is O(n) time, e.g. quick select; but it is not sorted, can be sorted in O(K log K) if needed. this approach runs in O(N) time, the constants hidden in the O-notation can be large. worst case of quick select is O(n^2)
- using min heap of size k (nlogk)
- using max heap of size n (n + klongn)

For non-streaming
approach 1:
Count freq + bucket sort
O(n), cons is the bucket is sparse, if there is **one extreme freq**, inefficient space usage

```java
public List<Integer> topKViewPhoto(int[] photoIds, int k) {
    List<Integer> result = new ArrayList<>();
    if(photoIds==null || photoIds.length<1) return result;

    Map<Integer, Integer> freqMap = new HashMap<>();
    int maxFreq = 0;
    for(int id: photoIds) {
        int freq = freqMap.getOrDefault(id, 0)+1;
        maxFreq = Math.max(maxFreq, freq);
        freqMap.put(id, freq);
    }

    List[] freqBuckets = new List[maxFreq+1];
    for(Map.Entry<Integer, Integer> freqEntry: freqMap.entrySet()) {
        if(freqBuckets[freqEntry.getValue()]==null) {
            freqBuckets[freqEntry.getValue()] = new ArrayList<>();
        }
        freqBuckets[freqEntry.getValue()].add(freqEntry.getKey());
    }

    for(int i=freqBuckets.length-1; i>=0; i--) {
        if(freqBuckets[i]!=null) {
            List<Integer> ids = freqBuckets[i];
            if(k>=freqBuckets[i].size()) {
                result.addAll(ids);
                k -= freqBuckets[i].size();
            } else {
                for(int j=0; j<k; j++) {
                    result.add(ids.get(j));
                }
                break;
            }
        }

    }

    return result;

}
```

approach 2:
using k-min heap to maintain k most viewed. nlogk

```
public List<Integer> topKViewPhoto(int[] photoIds, int k) {
    List<Integer> result = new ArrayList<>();
    if(photoIds==null || photoIds.length<1) return result;

    Map<Integer, Integer> freqMap = new HashMap<>();
    for(int id: photoIds) {
        int freq = freqMap.getOrDefault(id, 0)+1;
        freqMap.put(id, freq);
    }

    PriorityQueue<View> topKView = new PriorityQueue<>((a,b)->a.freq - b.freq);
    for(Map.Entry<Integer, Integer> freqEntry: freqMap.entrySet()) {
```

```java
            View view = new View(freqEntry.getKey(), freqEntry.getValue());
            if(topKView.size()<k) {
                topKView.add(view);
            } else {
                if(freqEntry.getValue()>topKView.peek().freq) {
                    topKView.poll();
                    topKView.offer(view);
                }
            }
        }

        while(!topKView.isEmpty()) {
            result.add(topKView.poll().id);
        }

        return result;

}

class View {
    int id;
    int freq;
    public View(int id, int freq) {
        this.id = id; this.freq = freq;
    }
}
```

For streaming:
The problem here is we cannot increase/ decrease value of priority queue, to remove it and re-add, the remove is O(k).
What we can do better is to implement the min heap ourselves, and do the keyAdjust heapify, which will be O(logk)
**check data structure heap

```java
class PhotoView implements Comparable<PhotoView> {
    int id;
    int freq;

    public PhotoView(int id, int freq) {
        this.id = id; this.freq = freq;
    }

    @Override
    public int compareTo(PhotoView other) {
        if(this.freq == other.freq) {
            return other.id - this.id;
        }
        return this.freq - other.freq;
    }
}
public class MaxPhotoID {
    private PriorityQueue<PhotoView> kMostViewPhotos;
    private Map<Integer, PhotoView> photoViewFreqMap;
    private final int k;

    public MaxPhotoID(int k) {
        this.k = k;
        kMostViewPhotos = new PriorityQueue<>();
        photoViewFreqMap = new HashMap<>();
    }

    public void view(int id) {
        if(!photoViewFreqMap.containsKey(id)) {
            photoViewFreqMap.put(id, new PhotoView(id, 0));
        }
        PhotoView view = photoViewFreqMap.get(id);
        view.freq++;

        if (kMostViewPhotos.size()<k ||
                view.freq >= kMostViewPhotos.peek().freq ) {
            kMostViewPhotos.remove(view);
            kMostViewPhotos.offer(view);
            if(kMostViewPhotos.size()>k)
                kMostViewPhotos.poll();
        }
    }

    public List<Integer> getTopKViewPhoto() {
        PhotoView[] topK = kMostViewPhotos.toArray(new PhotoView[kMostViewPhotos.size()]);
        /*Arrays.sort(topK, (a, b)-> { //cannot sort it
            if(a.freq == b.freq) {
                return a.id - b.id;
            }
            return a.freq - b.freq;
        });*/
        List<Integer> result = new ArrayList<>();
        for(PhotoView photoView: topK) {
            result.add(photoView.id);
        }
        return result;
    }

    public static void main (String[] args) {
        MaxPhotoID solver = new MaxPhotoID(4);
        solver.view(1);
        solver.view(2);
        solver.view(1);
        System.out.println(solver.getTopKViewPhoto());
        solver.view(3);

        System.out.println(solver.getTopKViewPhoto());
        solver.view(2);
        solver.view(12);
        solver.view(31);
        solver.view(101);
        solver.view(11);
        solver.view(3);
        System.out.println(solver.getTopKViewPhoto());

        solver.view(31);
        solver.view(101);
        solver.view(31);
        solver.view(101);
        System.out.println(solver.getTopKViewPhoto());

    }

}
```

Some people's idea about HashMap + DD Linked List: (it is just bubble sort, but with dynamic buckets)
view O(1); getTopK depends on the freq distribution, can be worst case O(k).
DD LL to maintain the buckets, if new bucket, just add it and update prev, next;; if after removing the element from old freq bucket, the bucket becomes empty, delete the node...

Phone Number & Dictionary

如果能得到词典的话。把词典的数据放到Trie tree里来达到自动过滤前缀的效果，每次DFS前用trie搜一下能减少DFS

Simple implementation:

```
public class PhoneNumberDict {

    private final static String[] KEYS = {"","","abc", "def", "ghi", "jkl", "mno", "pqrs", "tuv", "wxyz"};
    private static Set<String> dict;

    public List<String> letterCombinations(String digits) {
        List<String> combinations = new ArrayList<>();
        if(digits==null || digits.length()<1) return combinations;

        findCombinations(combinations, new StringBuilder(), digits, 0);
```

```java

        return combinations;
    }

    public void findCombinations(List<String> combinations, StringBuilder sb, String digits, int i) {
        if(i==digits.length()) {
            String comb = sb.toString();
            if(isWord(comb))
                combinations.add(comb);
        } else {
            String keyLetters = KEYS[digits.charAt(i)-'0'];
            for(int k=0; k<keyLetters.length(); k++) {
                sb.append(keyLetters.charAt(k));
                findCombinations(combinations, sb, digits, i+1);
                sb.setLength(sb.length()-1);
            }
        }
    }

    public boolean isWord(String word) {
        return dict.contains(word);
    }

    public static void main(String[] args) {
        dict = new HashSet<>();
        dict.add("drop");
        dict.add("box");
        dict.add("dropbox");

        List<String> combinations = new PhoneNumberDict().letterCombinations("3767269");
        for(String comb: combinations) {
            System.out.println(comb);
        }

    }

}
```

Follow up:

用TIRE当然不是说你已经凑好一个单词了，再去TRIE里搜索了，是带着TRIE NODE往下搜，遇到TRIE NODE CHILDREN里完全没有的，直接跳过，可以避免很多无意义的DFS，如果带着TRIE NODE往里搜，那么判断是不是单词也就O(1)，因为就CHECK一下那个ISWORD的BOOLEAN就行

Trie Tree

```java
class Trie {

    private class Node {
        Map<Character, Node> children;
        boolean isWord;

        public Node() {
            children = new HashMap<Character, Node>();
            isWord = false;
        }
    }

    private Node root;

    /** Initialize your data structure here. */
    public Trie() {
        root = new Node();
    }

    /** Inserts a word into the trie. */
    public void insert(String word) {
        Node curr = root;
        for(int i=0; i<word.length(); i++) {
            Node temp = curr.children.get(word.charAt(i));
            if(temp==null) {
                temp = new Node();
                curr.children.put(word.charAt(i), temp);
            }
            curr = temp;
        }
        curr.isCompleteWord = true;
    }

    /** Returns if the word is in the trie. */
    public boolean search(String word) {
        Node curr = root;
        for(int i=0; i<word.length(); i++) {
            curr = curr.children.get(word.charAt(i));
            if(curr==null) return false;
        }

        return curr.isWord;
    }

    /** Returns if there is any word in the trie that starts with the given prefix. */
    public boolean startsWith(String prefix) {
        Node curr = root;
        for(int i=0; i<prefix.length(); i++) {
            curr = curr.children.get(prefix.charAt(i));
            if(curr==null) return false;
        }
        return true;
    }
}
```

Game of Life

- performance bottleneck: disk read/write
- In distributed computing, can use MapReduce, key: top left and bottom right coordinates to represent the rectangle.

Memory full:

- linux cannot allocate memory, kill the current task
- Windows start use disk as virtual memory and speed down significantly

Linux commands to check memory:

```
top //check memory and cpu usage per process, but also reports total memory usage
free -m
```

Sharpness Values

Things to consider:

- Ask questions to clarify and let interviewer know you understand the problem before coding
- Explain the concept how it is DP = min(max(..,..,..), self)
- Space optimisation, tell the interviewer the observation it just depends on prev column result. we will do space opt later.

```
public class SharpnessValue {

    public int findSharpnessValue(int[][] matrix) {
        if(matrix==null || matrix.length<1 || matrix[0].length<1) return -1;

        int m = matrix.length, n = matrix[0].length;
        int[][] sharpness = new int[m][n];

        for(int i=0; i<m; i++) {
            sharpness[i][0] = matrix[i][0];
        }

        for(int j=1; j<n; j++) {
            for(int i=0; i<m; i++) {
              //find the max sharpness from the left, upper left, and lower left path
                int pathPrev = sharpness[i][j-1];
                if(i>0) {
                    pathPrev = Math.max(pathPrev, sharpness[i-1][j-1]);
                }
                if(i<m-1) {
                    pathPrev = Math.max(pathPrev, sharpness[i+1][j-1]);
                }

                sharpness[i][j] = Math.min(pathPrev, matrix[i][j]);
            }
        }

        int maxSharpness = Integer.MIN_VALUE;
        for(int i=0; i<m; i++) {
            maxSharpness = Math.max(maxSharpness, sharpness[i][n-1]);
        }

        return maxSharpness;

    }

    public static void main(String[] args) {
        int[][] matrix = {{1}, {2}, {3}};

        SharpnessValue solver = new SharpnessValue();
        assert solver.findSharpnessValue(matrix)==3;

        int[][] matrix2 = {{1,2,3}};
        assert solver.findSharpnessValue(matrix2)==1;

        int[][] matrix3 = {{3}};
        assert solver.findSharpnessValue(matrix3)==3;

        int[][] matrix4 = {{5,7,2},{7,5,8},{9,1,5}};
        System.out.println(solver.findSharpnessValue(matrix4));

    }
}
```

Space Optimised:

```
for(int j=1; j<n; j++) {
    int prev = sharpness[0];
    for(int i=0; i<m; i++) {
        //find the max sharpness from the left, upper left, and lower left path
        int maxPath = sharpness[i];
        if(i>0) {
            maxPath = Math.max(maxPath, prev);
        }
        if(i<m-1) {
            maxPath = Math.max(maxPath, sharpness[i+1]);
        }

        prev = sharpness[i];
        sharpness[i] = Math.min(maxPath, matrix[i][j]);
    }
}
```

Follow-up:
If the matrix is very large:
*cannot process rectangle by rectangle, because the boundary of the rect depends on the next row of the prev, and the next row prev is depends on the doundary of ... recursive...!
*The same reason, cannot do it 3 rows by 3 rows each time.

1. Read it column by column each time (many disk seek())
2. Transpose the file: same if read row, output col, many disk seek() when write; if read col, output row, many disk seek() when read
    - we can according to the memory size, each time read a square matrix, and do the transpose of it. (need RandomAccessFile, seek() etc)
    - Then do the processing row by row.

Buy Soda/ Coin Change - Combination Sum

```
Given a set of candidate numbers (C) (without duplicates) and a target number (T), find all unique combinations in C where the candidate numbers sum
```

```
s to T.
The same repeated number may be chosen from C unlimited number of times.
Note:
    - All numbers (including target) will be positive integers.
    - The solution set must not contain duplicate combinations.
For example, given candidate set [2, 3, 6, 7] and target 7,
A solution set is:
[
  [7],
  [2, 2, 3]
]
```

Recursive:
complexity在 (sizes.length)^n 级别，因为n可以非常大，所以很容易指数爆炸，但是一般来说pack sizes只有那么几种，所以她想让我写 n^(sizes.length) 级别的解法

Recursive time complexity: each level there is a loop, the branching factor could up to m, the depth could up to O(T), so M^T (*it is not 2^n as the element can be used more than one, if the element can use only once, actually we are considering all combination, 2^n such combinations)
Space: just use the result array and one list for backtracking.

```
class Solution {
    public List<List<Integer>> combinationSum(int[] candidates, int target) {
        List<List<Integer>> result = new ArrayList<>();
        Arrays.sort(candidates);
        combinationSum(result, new ArrayList(), candidates, target, 0);
        return result;
    }

    public void combinationSum(List<List<Integer>> result, List<Integer> list, int[] candidates, int target, int index) {
        if(target<0) return;
        if(target==0) {
            result.add(new ArrayList<Integer>(list)); return;
        }

        for(int i=index; i<candidates.length && candidates[i]<=target; i++) {
            list.add(candidates[i]);
            combinationSum(result, list, candidates, target-candidates[i], i);
            list.remove(list.size()-1);
        }
    }
}
```

DP:

Time complexity: T*mk, where k is the partial solution set size*

Space: use up much more space as we stored so many partial solutions

```
class Solution {
    public List<List<Integer>> combinationSum(int[] candidates, int target) {
        List<List<List<Integer>>> dp = new ArrayList<>();//be careful, 0 base index
        Arrays.sort(candidates);

        for(int t=1; t<=target; t++) {
            ArrayList<List<Integer>> newList = new ArrayList<>();
            for(int i=0; i<candidates.length && candidates[i]<=t; i++) {
                if(t==candidates[i]) {
                    newList.add(Arrays.asList(candidates[i]));
                } else {
                    for(List<Integer> list: dp.get(t-candidates[i]-1)) {
                        //this is to pick the list seq which is monotonic increasing
                        //this can avoid duplicates
                        if(candidates[i]>=list.get(0)) {
                            List<Integer> cumList = new ArrayList<>();
                            cumList.add(candidates[i]); cumList.addAll(list);
                            newList.add(cumList);
                        }
                    }
                }
            }
            dp.add(newList);
        }

        return dp.get(target-1);
    }
}
```

Word Pattern

Given pattern and a list of words:

- bijection mapping, so need to check if the word is already mapped.

```
class Solution {
    public boolean wordPattern(String pattern, String str) {
        String[] patternStrMap = new String[258];
        Set<String> mapped = new HashSet<>();

        String[] words = str.split(" ");
        if(pattern.length()!=words.length)
            return false;

        for(int i=0; i<pattern.length(); i++) {
            char c = pattern.charAt(i);
            if(patternStrMap[c]==null) {
                if(mapped.contains(words[i]))
                    return false;
                patternStrMap[c] = words[i];
                mapped.add(words[i]);
            } else {
                if(!patternStrMap[c].equals(words[i]))
                    return false;
            }
        }

        return true;
    }
}
```

The given input is not a list of words

```
public class WordPatternMatch {
    public boolean wordPatternMatch(String pattern, String str) {
        Map<Character, String> charStrMap = new HashMap<>();
        Set<String> usedStr = new HashSet<>();
        return isMatch(charStrMap, usedStr, pattern, 0, str, 0);
    }

    public boolean isMatch(Map<Character, String> charStrMap, Set<String> usedStr, String pattern, int i, String str, int j) {
        //System.out.println(i + "  "+j);
        if(pattern.length()==i && str.length()==j) return true;
        if(pattern.length()==i || str.length()==j) return false;

        char c = pattern.charAt(i);
        if(charStrMap.containsKey(c)) {
            String mappedStr = charStrMap.get(c);
            if(str.startsWith(mappedStr, j)) {
                return isMatch(charStrMap, usedStr, pattern, i+1, str, j+mappedStr.length());
            } else {
                return false;
            }
        } else {
            for(int k=j; k<str.length(); k++) {
                String word = str.substring(j, k+1);
                //System.out.println(word);
                if(usedStr.contains(word)) continue;
                usedStr.add(word);
```

```
                    charStrMap.put(c, word);
                    if(isMatch(charStrMap, usedStr, pattern, i+1, str, k+1))
                        return true;
                    usedStr.remove(word);
                    charStrMap.remove(c);
                }
                return false;
            }

        }

    public static void main(String[] args) {
        System.out.println(new WordPatternMatch().wordPatternMatch("abba", "dogcatcatdog"));
    }
}

```

Word Break /Word Break II

```
Given a **non-empty** string *s* and a dictionary *wordDict* containing a list of **non-empty** words, determine if *s* can be segmented into a space-separated sequence of one or more dictionary words. You may assume the dictionary does not contain duplicate words.
For example, given
s = "leetcode",
dict = ["leet", "code"].
Return true because "leetcode" can be segmented as "leet code".
```

```
class Solution {
    public boolean wordBreak(String s, List<String> wordDict) {
        boolean[] dp = new boolean[s.length()+1];
        dp[0]=true;

        for(int i=1; i<=s.length(); i++) {
            for(int j=0; j<i; j++) {
                if(dp[j] && wordDict.contains(s.substring(j, i))) {
                    dp[i] = true; break; //here dp[i] refers to prev 0..i-1 chars can be wordbreak
                }

            }
        }

        return dp[s.length()];
    }
}
```

If we need to output the solution:

```
class Solution {
    public List<String> wordBreak(String s, List<String> wordDict) {
        Map<String, List<String>> lookup = new HashMap<>();
        return wordBreak(s, wordDict, lookup);
    }

    public List<String> wordBreak(String s, List<String> wordDict,
                                  Map<String, List<String>> lookup) {
        if(lookup.get(s)!=null) return lookup.get(s);

        List<String> result = new ArrayList<>();
        if(s.length()==0) {
            result.add("");
            return result;
        }

        for(String word: wordDict) {
            if(s.startsWith(word)) {
                List<String> partialResult = wordBreak(s.substring(word.length()) ,
                            wordDict, lookup);
                for(String str: partialResult) {
                    result.add(word + (str.equals("")? "":" "+ str));
                }
            }
        }
        lookup.put(s, result);

        return result;
    }

}
```

Number of Islands

Given a 2d grid map of `'1'`s (land) and `'0'`s (water), count the number of islands. An island is surrounded by water and is formed by connecting adjacent lands horizontally or vertically. You may assume all four edges of the grid are all surrounded by water.

Example 1:

```
11110
11010
11000
00000
```

```
class Solution {
    public static final char ISLAND_MARK = '1';
    public static final char WATER_MARK = '0';

    public int numIslands(char[][] grid) {
        int counter = 0;
        if(grid==null || grid.length<1) return counter;
        for(int i=0; i<grid.length; i++) {
            for(int j=0; j<grid[i].length; j++) {
                if(grid[i][j]==ISLAND_MARK) {
                    counter++;
                    mark(grid, i,j);
                }
            }
        }
        return counter;
    }

    public void mark(char[][] grid, int i, int j) {
        if(i>=0 && i<grid.length && j>=0 && j<grid[i].length && grid[i][j]
== ISLAND_MARK) {
            grid[i][j] = WATER_MARK;
            mark(grid, i-1, j);
            mark(grid, i+1, j);
            mark(grid, i, j-1);
            mark(grid, i, j+1);
        }
    }
}
```

If file very large, read row by row, using Union Find:

305. Number of Islands II

A 2d grid map of `m` rows and `n` columns is initially filled with water. We may perform an *addLand* operation which turns the water at position (row, col) into a land. Given a list of positions to operate, **count the number of islands after each *addLand* operation.** An island is surrounded by water and is formed by connecting adjacent lands horizontally or vertically. You may assume all four edges of the grid are all surrounded by water.
Example:
Given `m = 3, n = 3`, `positions = [[0,0], [0,1], [1,2], [2,1]]`.
Initially, the 2d grid `grid` is filled with water. (Assume 0 represents water and 1 represents land).

```
0 0 0
0 0 0
0 0 0
```

Operation #1: addLand(0, 0) turns the water at grid[0][0] into a land.

```
1 0 0
0 0 0   Number of islands = 1
0 0 0
```

```java
class Solution {
    private static final int[][] directions = {{0, 1}, {1, 0}, {-1, 0}, {0, -1}};

    public List<Integer> numIslands2(int m, int n, int[][] positions) {
        List<Integer> result = new ArrayList<>();
        if(m==0 || n==0 || positions==null || positions.length<1) return result;

        int[] root = new int[m*n]; //id = n*i+j
        int[] rank = new int[m*n];
        Arrays.fill(root, -1);

        int numIslands = 0;
        for(int[] position: positions) {
            int id = position[0]*n + position[1];
            root[id] = id; rank[id] = 1;
            numIslands++;
            for(int k=0; k<directions.length; k++) {
                int i = position[0] + directions[k][0];
                int j = position[1] + directions[k][1];

                if(i>=0 && i<m && j>=0 && j<n && root[i*n+j]!=-1) {
                    numIslands = union(root, rank, id, i*n+j, numIslands);
                }

            }

            result.add(numIslands);
        }

        return result;
    }

    public int union(int[] root, int[] rank, int x, int y, int numIslands)
{
        int parent1 = find(root, x);
        int parent2 = find(root, y);
        if(parent1!=parent2) { //union
            numIslands--;
            if(rank[parent1]< rank[parent2]) {
                root[parent1] = parent2;
                rank[parent2] += rank[parent1];
            } else {
                root[parent2] = parent1;
                rank[parent1] += rank[parent2];
            }
        }
        return numIslands;
    }

    public int find(int[] root, int id) {
        while(root[id]!=id) {
            root[id] = root[root[id]];
            id = root[id];
        }
        return root[id];
    }

}
```

Folder Access

Given child-parent folder relationships, and user has access to folders in the access set. Find if user has access to particular folder.

```
/A
|_ _/B
    |__/C <-- access
    |__/D
|__/E <-- access
    |__/F
```

|__ /G
folders = [[br] ('A', None),
('B', 'A'),
('C', 'B'),
('D', 'B'),
('E', 'A'),
('F', 'E'),
]
access = set(['C', 'E'])
has_access(String folder_name) -> boolean
has_access("B") -> false
has_access("C") -> true
has_access("F") -> true
~~has_access("G") -> true (this is probably wrong)~~

Things to consider:

- The child-parent relationships are given in HashMap?
- can a folder has more than one parent? If so, how is it represented?

Linux command to create symbolic link to folder:

```
ln -s path-to-actual-folder name-of-link
ls -ld name-of-link
```

- Would it be possible a circular child-parent relationship? if so, mark visited.
- Need to return false if you are checking a folder non-exiting.

```java
package DropBox;
import java.util.*;

public class FolderAccess {
    private Map<String, String> foldersParent;
    private Set<String> access;

    public FolderAccess(Map<String, String> foldersParent, Set<String> access) {
        this.foldersParent = foldersParent;
        this.access = access;
    }

    public boolean hasAccess(String folderName) {
        String currFolder = folderName;
        while(currFolder!=null) {
            if(access.contains(currFolder)) {
                return true;
            } else {
                currFolder = foldersParent.get(currFolder);
            }
        }
        return false;
    }

    public static void main(String[] args) {
        Map<String, String> foldersParent = new HashMap<>();
        foldersParent.put("B", "A");
        foldersParent.put("C", "B");
        foldersParent.put("D", "B");
        foldersParent.put("E", "A");
        foldersParent.put("F", "E");

        Set<String> access = new HashSet<>();
        access.add("C");
        access.add("E");

        FolderAccess solver = new FolderAccess(foldersParent, access);
        assert solver.hasAccess("B") == false;
        assert solver.hasAccess("C") == true;
        assert solver.hasAccess("F") == true;
        assert solver.hasAccess("G") == false; //G is not in the folders structure
    }
}

```

Count And Say

```
class Solution {
    public String countAndSay(int n) {
        String curr = "1";
        StringBuilder sb = new StringBuilder();
        for(int i=2; i<=n; i++) {
            sb.setLength(0);
            int count = 1;
            for(int j=0; j<curr.length(); j++) {
                if(j==curr.length()-1 || curr.charAt(j)!=curr.charAt(j+1))
{
                    sb.append(count).append(curr.charAt(j));
                    count=1;
                } else
                    count++;
            }
            curr = sb.toString();
        }

        return curr;
    }
}
```

Folders And Cows

Grid Illumination

Multi-threading

Web Crawler

http://scrumbucket.org/tutorials/neo4j-site-crawler/part-2-create-multi-threaded-crawl-manager/

Multithreading Things to consider:

1. which part is most time-consuming. The part that the thread knows the url to be visited and getting back list of URLs. Network latency, webpage content parser and processing.
2. Should visit URL? e.g. define the depth of crawling, type of urls e.g. the one without ending in .pdf, size of the result set etc.
3. Crawler failed, will throws ExecutionException
4. Sleep the master thread a little bit each time after the checking of futures... manager thread will not use all the resources

Simple single thread DFS, BFS

```
public List<String> crawlBFS(String url) {
    Set<String> result = new HashSet<>();
    Queue<String> q = new LinkedList<>();
    q.add(url); result.add(url);

    while(!q.isEmpty()) {
        String currUrl = q.poll();

        List<String> childrenUrls = getUrls(currUrl);
        if(childrenUrls!=null) {
            for(String childUrl: childrenUrls) {
                if(!result.contains(childUrl)) {
                    q.offer(childUrl);
                    result.add(childUrl);
                }
            }
        }
    }

    return new ArrayList<>(result);
}

public List<String> crawlDFS(String url) {
    Set<String> result = new HashSet<>();

    crawlDFSHelper(result, url);

    return new ArrayList<>(result);
}

public void crawlDFSHelper(Set<String> result, String url) {
    if(url==null || result.contains(url)) return;

    result.add(url);
    List<String> childrenUrls = getUrls(url);
    if(childrenUrls!=null) {
        for(String childUrl: childrenUrls) {
            crawlDFSHelper(result, childUrl);
        }
    }

}
```

Multithread version:

1. since the most time consuming part is the getUrls(url) function, we can use a Master/slave approach to parallel processing the getUrls(url),
2. The benefit of also making the read write of result set parallel doesn't justify synch overhead. So, prefer just the master thread to do it.

```

import java.util.*;
import java.util.concurrent.*;

/*
    http://scrumbucket.org/tutorials/neo4j-site-crawler/part-2-create-multi-threaded-crawl-manager/
 */
public class CrawlerManager {
    public static final int THREAD_COUNT = 10;
    private static final long PAUSE_TIME = 1000;

    private Set<String> result = new HashSet<>();
    private List<Future<List<String>>> futures = new ArrayList<>();
    private ExecutorService executor = Executors.newFixedThreadPool(THREAD_COUNT);

    private static Map<String, List<String>> connectedUrls;


    public static List<String> getUrls(String url) {
        return connectedUrls.getOrDefault(url, new ArrayList<String>());
    }

    public List<String> crawl(String startUrl) {
        submitNewUrl(startUrl);

        try{
            while(checkCrawlerResults());
        } catch (InterruptedException e) {

        }
        executor.shutdown();
        return new ArrayList<>(result);
    }

    public boolean checkCrawlerResults() throws InterruptedException {
        Thread.sleep(PAUSE_TIME);
        Iterator<Future<List<String>>> iterator = futures.iterator();
        List<String> newUrls = new ArrayList<>();

        while(iterator.hasNext()) {
            Future<List<String>> future = iterator.next();
            if(future.isDone()) {
                iterator.remove();
                try {
                    newUrls.addAll(future.get());
                } catch (ExecutionException e) {

                }
            }
        }

        // do this after the while iterator because submitNewUrl will change the futures array list,
        // will cause concurrent modification error
        for(String url: newUrls) {
            submitNewUrl(url);
        }

        return futures.size()>0;
    }

    public void submitNewUrl(String url) {
        if(!result.contains(url)) {
            result.add(url);

            Crawler crawler = new Crawler(url);
            Future<List<String>> future = executor.submit(crawler);
            futures.add(future);
        }
    }

    private class Crawler implements Callable<List<String>> {
        private String url;

        public Crawler(String url) {
            this.url = url;
        }

        @Override
        public List<String> call() {
            List<String> urls = getUrls(url);
            return urls;
        }
    }

    public static void main(String[] args) {
        connectedUrls = new HashMap<>();
        List<String> aChildren = new ArrayList<>();
        aChildren.add("b");
        aChildren.add("c");
        aChildren.add("d");
        aChildren.add("e");
        List<String> bChildren = new ArrayList<>();
        bChildren.add("k");
        bChildren.add("m");
        bChildren.add("d");
        bChildren.add("z");
        List<String> kChildren = new ArrayList<>();
        kChildren.add("o");
        kChildren.add("j");
        kChildren.add("e");
        kChildren.add("z");

        connectedUrls.put("a", aChildren);
        connectedUrls.put("b", bChildren);
        connectedUrls.put("k", kChildren);

        CrawlerManager crawlerManager = new CrawlerManager();
        System.out.println(crawlerManager.crawl("a"));

    }
}
```

How the urls are gathered:

```
public List<String> getUrls(String urlStr) {
        List<URL> urlList = new ArrayList<>();

        URL url = new URL(urlStr);
        Document doc = Jsoup.parse(url, TIMEOUT);

        Elements links = document.select("a[href]");
        for(Element link: links) {
                String href = link.attr("href");
                if(StringUtils.isBlank(href) || href.startsWith("#")) {
                        continue;
                }
                try {
                        URL nextUrl = new URL(url, href);
                        urlList.add(nextUrl);
                } catch (MalformedURLException e) {

                }
        }

        return urlList;

}
```

Read Write Lock

http://tutorials.jenkov.com/java-concurrency/read-write-locks.html
#java_lock

- Grant Read access when there is no writers (can be many readers at the same time)
- Grant Write access when there is no readers and no writers (only one writer can write at the same time)

Why notifyAll()?

- all threads waiting for read access are granted read access at once - not one by one.
- Inside the ReadWriteLock there are threads waiting for read access, and threads waiting for write access. If a thread awakened by notify() was a read access thread, it would be put back to waiting because there are threads waiting for write access. However, none of the threads awaiting write access are awakened, so nothing more happens. No threads gain neither read nor write access. By calling noftifyAll() all waiting threads are awakened and check if they can get the desired access.

Starvation

Consider the situation more write or more read, we need to prioritise the read/write to avoid starvation. To do this, we can add additional count call writeRequests/readRequest. If read/write more or less the same, no need prioritise

Simple Version w/o Reentrant

```
public class ReadWriteLock{
  private int readers       = 0;
  private int writers       = 0;
  private int writeRequests = 0;

  public synchronized void lockRead() throws InterruptedException{
    while(writers > 0 || writeRequests > 0){
      wait();
    }
    readers++;
  }

  public synchronized void unlockRead(){
    readers--;
    notifyAll();
  }

  public synchronized void lockWrite() throws InterruptedException{
    writeRequests++;
    while(readers > 0 || writers > 0){
      wait();
    }
    writeRequests--;
    writers++;
  }

  public synchronized void unlockWrite() throws InterruptedException{
    writers--;
    notifyAll();
  }
}
```

Reentrant
If no reentrant and the writer tries to acquire read access before releasing write access, will cause infinite lock. 😖
Read -> Read
The thread holding the read access should be able to reenter if there is no writer writing.
Write -> Write
Currently holding write lock
Read -> write
if it is **the only reader**
Write -> read
should always granted for writer

```java
public class ReadWriteLock{

  private Map<Thread, Integer> readingThreads =
       new HashMap<Thread, Integer>();

  private int writeAccesses    = 0;
  private int writeRequests    = 0;
  private Thread writingThread = null;

  public synchronized void lockRead() throws InterruptedException{
    Thread callingThread = Thread.currentThread();
    while(! canGrantReadAccess(callingThread)){
      wait();
    }

    readingThreads.put(callingThread,
     (getReadAccessCount(callingThread) + 1));
  }

  private boolean canGrantReadAccess(Thread callingThread){
    if( isWriter(callingThread) ) return true;
    if( hasWriter()             ) return false;
    if( isReader(callingThread) ) return true;
    if( hasWriteRequests()      ) return false;
    return true;
  }

  public synchronized void unlockRead(){
    Thread callingThread = Thread.currentThread();
    if(!isReader(callingThread)){
      throw new IllegalMonitorStateException("Calling Thread does not" +
        " hold a read lock on this ReadWriteLock");
    }
    int accessCount = getReadAccessCount(callingThread);
    if(accessCount == 1){ readingThreads.remove(callingThread); }
    else { readingThreads.put(callingThread, (accessCount -1)); }
    notifyAll();
  }

  public synchronized void lockWrite() throws InterruptedException{
    writeRequests++;
    Thread callingThread = Thread.currentThread();
    while(! canGrantWriteAccess(callingThread)){
      wait();
    }
    writeRequests--;
    writeAccesses++;
    writingThread = callingThread;
  }

  public synchronized void unlockWrite() throws InterruptedException{
    if(!isWriter(Thread.currentThread()){
      throw new IllegalMonitorStateException("Calling Thread does not" +
        " hold the write lock on this ReadWriteLock");
    }
    writeAccesses--;
    if(writeAccesses == 0){
      writingThread = null;
    }
    notifyAll();
  }

  private boolean canGrantWriteAccess(Thread callingThread){
    if(isOnlyReader(callingThread))    return true;
    if(hasReaders())                   return false;
    if(writingThread == null)          return true;
    if(!isWriter(callingThread))       return false;
    return true;
  }

  private int getReadAccessCount(Thread callingThread){
    Integer accessCount = readingThreads.get(callingThread);
    if(accessCount == null) return 0;
    return accessCount.intValue();
  }

  private boolean hasReaders(){
    return readingThreads.size() > 0;
  }

  private boolean isReader(Thread callingThread){
    return readingThreads.get(callingThread) != null;
  }

  private boolean isOnlyReader(Thread callingThread){
    return readingThreads.size() == 1 &&
           readingThreads.get(callingThread) != null;
  }

  private boolean hasWriter(){
    return writingThread != null;
  }

  private boolean isWriter(Thread callingThread){
    return writingThread == callingThread;
  }

  private boolean hasWriteRequests(){
      return this.writeRequests > 0;
  }
}
```

- If the code might cause Exceptions, **put .unlock in the finally block.**

Token Bucket

```
public class TokenBucket {
    private final int MAX_CAPACITY;
    private final int FILL_RATE;

    private List<Integer> bucket;
    private long lastFillTimeStamp;

    final Lock lock = new ReentrantLock();
    final Condition notFull = lock.newCondition();
    final Condition notEmpty = lock.newCondition();

    public TokenBucket(int maxCapacity, int fillRate) {
        this.MAX_CAPACITY = maxCapacity;
        this.FILL_RATE = fillRate;
        lastFillTimeStamp = System.currentTimeMillis();

        bucket = new ArrayList<>();
    }

    public void fill() throws InterruptedException {
        lock.lock();
        while (bucket.size() == MAX_CAPACITY) {
            System.out.println("Bucket is filled now.");
            notFull.await();
        }
        long now = System.currentTimeMillis();
        //System.out.println((now - lastFillTimeStamp)/1000 * FILL_RATE);
        long numTokensToAdd = Math.min(MAX_CAPACITY - bucket.size(), (now - lastFillTimeStamp)/1000 * FILL_RATE);
        lastFillTimeStamp = now;
        //System.out.println(numTokensToAdd);
        for(int i=0; i<numTokensToAdd; i++) { //add tokens
            bucket.add((int) (Math.random() * 100) + 1);
        }
        notEmpty.signalAll();
        lock.unlock();

    }

    public List<Integer> get(int n) throws InterruptedException{
        //System.out.println(n);
        if(n<=0)
            throw new IllegalArgumentException("Cannot get zero or negative number of tokens.");
        if(n>MAX_CAPACITY)
            throw new IllegalArgumentException("Cannot get tokens more than max capacity.");

        List<Integer> result = new ArrayList<>();

        int tokenAcquired = 0;
        while(tokenAcquired<n) { //this can ensure fair competition
            lock.lock();
            while(bucket.size()<1) {
                System.out.println("Bucket is not enough now.");
                notEmpty.await();
            }
            result.add(bucket.get(bucket.size()-1));
            bucket.remove(bucket.size()-1);
            tokenAcquired++;
            notFull.signalAll();
            lock.unlock();

        }

        return result;
    }

    public static void main(String[] args) {
        TokenBucket bucket = new TokenBucket(100, 8);
        Runnable filler = () -> {
            while(true) {
                try {
                    bucket.fill();
                    Thread.sleep(1000);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        };

        Runnable consumer = () -> {
            while(true) {
                try {
                    List<Integer> result = bucket.get((int) (Math.random() * 5) + 1);
                    System.out.println(result.toString());
                    Thread.sleep(3000);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        };

        new Thread(filler).start();
        new Thread(consumer).start();
    }

}
```

Design

Hit Counter

Things to consider:

- If a lot of hits within given time window / per sec
- If it keeps hit() for a long time only then call getHit(), memory problem (clear outdated also when hit() is called)
- Multi-threading, if multiple threads report hit() // if a webpages, multiple clients click the hit

Simple version using queue:
Space: varies, depends on number of hits in given time window
Time: varies, depends on number of outdated hits

```
public class HitCounter {
  private static final int WINDOW = 5*60; // 5 minutes
  private Queue<Integer> timeStamps;

  public HitCounter() {
    timeStamps =  new LinkedList<>();
  }

  public void hit() {
    int currTime = getCurrTimeSec();
    timeStamps.offer(currTime);
  }

  public void getHits() {
    int currTime = getCurrTimeSec();
    while(!timeStamps.isEmpty() && timeStamps.peek() <= currTime - WINDOW)
{
      timeStamps.poll();
    }
    return timeStamps.size();
  }

  public int getCurrTimeSec() {
    return System.currTimeMillis()/1000;
  }
}
```

Circular Array approach:

- think about it as assigning the hits to buckets, 300 buckets, next time you put hits into the bucket, the previous hits are already outdated, you can safely overwrite it.

Space: O(WINDOW_LENGTH)
Time: O(WINDOW_LENGTH)

```
//circular array

//|--------------------|--------------------|
public class HitCounter {
  private final int WINDOW_LENGTH; // in sec
  private Hit[] hitRecords;

  public HitCounter(int windowLength) {
    this.WINDOW_LENGTH = windowLength;
    hitRecords = new Hit[windowLength];
  }

  public void hit() {
    int currTime = getCurrTimeSec();
    int index = currTime % WINDOW_LENGTH;
    if(hitRecords[index]!=null && hitRecords[index].timeStamp==currTime) {
      hitRecords[index].count++;
    } else {
      hitRecords[index] = new Hit(currTime, 1);
    }
  }

  public int getHits() {
    int currTime = getCurrTimeSec();
    int count=0;
    for(Hit hit: hitRecords) {
      if (hit!=null && hit.timeStamp>currTime-WINDOW_LENGTH)
        count += hit.count;
    }
    return count;
  }

  public int getCurrTimeSec() {
    return (int)System.currentTimeMillis()/1000;
  }

  private class Hit {
    int timeStamp;
    int count;

    public Hit(int timeStamp, int count) {
      this.timeStamp = timeStamp;
      this.count = count;
    }
  }
}
```

We can avoid storing the timestamp attribute by storing the lastHitTime, and clear those since lastHitTime till currentHitTime / getHitTime to avoid invalid counts.

```
public void clearBucketFromLastHit(int currTime) {
    if(lastHitTime!=-1) {
        for(int i=lastHitTime+1; i<=currTime; i++) {
            hitRecords[i%WINDOW_LENGTH] = 0;
        }
    }
}
```

Multi-threading:

1. use BlockingQueue, **Linked**BlockingQueue is not bounded
2. synchronized the modification and reading from queue
3. Use ConcurrentHashMap, **it is not blocking on the whole collection when write**, so can have multiple writes at the same time, read is non blocking, using volatile... This is achieved by partitioning Map into different parts based on concurrency level and locking only a portion of Map during updates.

Testing:

1. Use Thread.sleep() to wait which is stupid! can change the methods to accept timestamps.
2. Use mockito to mock the System.currentTimeMillis().

Test cases:

1. No hits within pass 300 sec
2. Boundary case getHits()
3. Multiple hits within a sec
4. Consecutive getHits()

counter.getHits(0);
counter.hit(1);
counter.hit(2);
counter.hit(2);
counter.hit(2);
counter.hit(300);
counter.getHits(300);
counter.getHits(300);
counter.getHits(301);

Space Panorama

```
import java.io.*;
import java.util.*;

/**
* NASA selects Dropbox as its official partner, and we're tasked with managing
* a panorama for the universe. The Hubble telescope (or some other voyager we
* have out there) will occasionally snap a photo of a sector of the universe,
* and transmit it to us. You are to help write a data structure to manage this.
* For the purpose of this problem, assume that the observable universe has been .
* divided into 2D sectors. Sectors are indexed by x- and y-coordinates.
*/
public File {
    public File(String path) {}
    public Boolean exists() {}
    public byte[] read() {}
    public void write(bytes[] bytes) {}
}

public Image {
    public Image(byte[] bytes) {}
    byte[] getBytes() {} // no more than 1MB in size
}

public Sector {
    public Sector(int x, int y) {}
    int getX() {}
    int getY() {}

    @Override
    public boolean equals(Object o) {
        if(o==this) return true;

        if(!(o instanceof Sector)){
            return false;
        }

        Sector that = (Sector) o;

        return this.x == that.getX() && this.y == that.getY();
    }

    @Override
    public int hashCode() {
        int prime = 31;
        int result = 1;
        result = prime*result + this.x;
        result = prime*result + this.y;
        return result;
    }
}

/**
* row-major indexing to be consistent.
*/
public class SpacePanorama {
    /**
     * initializes the data structure. rows x cols is the sector layout.
     * width, height can be as large as 1K each.
     */
    public SpacePanorama(int rows, int cols) {}

    /**
     * The Hubble will occasionally call this (via some radio wave communication).
     * to report new imagery for the sector at (y, x).
     * Images can be up to 1MB in size.
     */
    public void update(int y, int x, Image image) {}

    /**
     * NASA will occasionally call this to check the view of a particular sector.
     */
    public Image fetch(int y, int x) {}

    /**
     * return the 2D index of the sector that has the stalest data.
     * the idea is that this may help the telescope decide where to aim ne
xt..
     */
    public Sector getStalestSector() {} //use LinkedHashSet??

    public void removeHead() {
        locMap.put(head.next.key, null);
        head.next = head.next.next;
        head.next.prev = head;
    }

    public void addTail(int key, int value) {
        DLinkedList newEle = new DLinkedList(key, value);
        moveToTail(newEle);
        locMap.put(key, newEle);
    }

    public void moveToTail(DLinkedList e) {
        e.prev = tail.prev;
        tail.prev.next = e;
        tail.prev = e;
        e.next = tail;
    }
}
```

LRU

```
class LRUCache {
    private int capacity;
    private LinkedHashMap<Integer, Integer> leastRecentUsedList;

    public LRUCache(int capacity) {
        this.capacity = capacity;
        leastRecentUsedList = new LinkedHashMap<>();
    }

    public int get(int key) {
        if(leastRecentUsedList.containsKey(key)) {
            int value = leastRecentUsedList.get(key);
            leastRecentUsedList.remove(key);
            leastRecentUsedList.put(key, value);
            return value;
        }

        return -1;

    }

    public void put(int key, int value) {
        if(leastRecentUsedList.containsKey(key)) {
            leastRecentUsedList.remove(key);
        } else if(leastRecentUsedList.size()==capacity) {
            leastRecentUsedList.remove(leastRecentUsedList.keySet().iterator().next());
        }
        leastRecentUsedList.put(key, value);
    }


}
/*class LRUCache {
    class DLinkedList {
        int key, val;
        DLinkedList prev, next;

        public DLinkedList(int _key, int _val) {
            key = _key; val = _val;
        }
    }

    private int capacity, count = 0;
    private DLinkedList head, tail;
    private Map<Integer, DLinkedList> locMap = new HashMap<Integer, DLinke
dList>();

    public LRUCache(int capacity) {
        this.capacity = capacity;
        head = new DLinkedList(-1, -1);//dummy for easier pointer manipula
tion
        tail = new DLinkedList(-1, -1);
        head.next = tail;
        tail.prev = head;
    }

    public int get(int key) {
        DLinkedList e = getNode(key);
        return  e == null? -1: e.val;
    }

    public DLinkedList getNode(int key) {
        if(locMap.get(key)!=null) {
            DLinkedList e = locMap.get(key);
            e.next.prev = e.prev;
            e.prev.next = e.next;
            moveToTail(e);
            return e;
        } else {
            return null;
        }
    }

    public void put(int key, int value) {
        DLinkedList e = getNode(key);
        if(e!=null) {
           e.val = value;
        } else {
            if(count==capacity) { // remove LRU
                removeHead();
            }
            addTail(key, value);
            count++;
        }

    }

    public void removeHead() {
        locMap.put(head.next.key, null);
        head.next = head.next.next;
        head.next.prev = head;
        count--;
    }

    public void addTail(int key, int value) {
        DLinkedList newEle = new DLinkedList(key, value);
        moveToTail(newEle);
        locMap.put(key, newEle);
    }

    public void moveToTail(DLinkedList e) {
        e.prev = tail.prev;
        tail.prev.next = e;
        tail.prev = e;
        e.next = tail;
    }
}*/

/**
```

```
 * Your LRUCache object will be instantiated and called as such:
 * LRUCache obj = new LRUCache(capacity);
 * int param_1 = obj.get(key);
 * obj.put(key,value);
 */
```

KV Store Transaction

Design Logging System (Distributed)

Design Twitter

Upload File to S3 & Thumbnail