Project 4: Heaps

Background:

Pin numbers, much like every other human generated value, are not in the least bit random. Typically speaking, it is much easier to “hack” a person than it is to hack a system or its hardware. In cyber security, this is referred to as targeting the “wetware” as opposed to targeting the software or hardware. Rather than guessing or brute forcing through pins randomly, we can seed our algorithm with popular pins.

In 2013, a blogger posted an analysis of credit card pins showing that approximately **50%** of user pins are covered within **400** unique numbers (4% of available pins). Conversely, the last 10% of passwords are spread amongst 4,000 unique numbers. This would imply that you could easily guess more than half user pins with 96% less effort than a standard brute force attack.

As it turns out, this problem isn’t just limited to pins, but to all facets of cyber security and human (patterns of) life. Given any large dataset of ‘human generated’ data, you should expect to be able to pick apart habits and casual relationships.

For this project, you will design min and max heaps that will make sorting and accessing high priority data efficient. In our case, we want to try to guess the most commonly used pins/passwords in order to get the most likely candidate as soon as possible by using a max heap. Statistically speaking, the most common pins will be the most likely to be encountered, and should show during your calls to ‘hack’. Conversely, you will also try your hand using the least popular pins by implementing a min heap and trying them in order. For reference, in this assignment, there is a ~10% chance of randomly succeeding with a marble bag full of all pins, removing them without replacement.

Goal

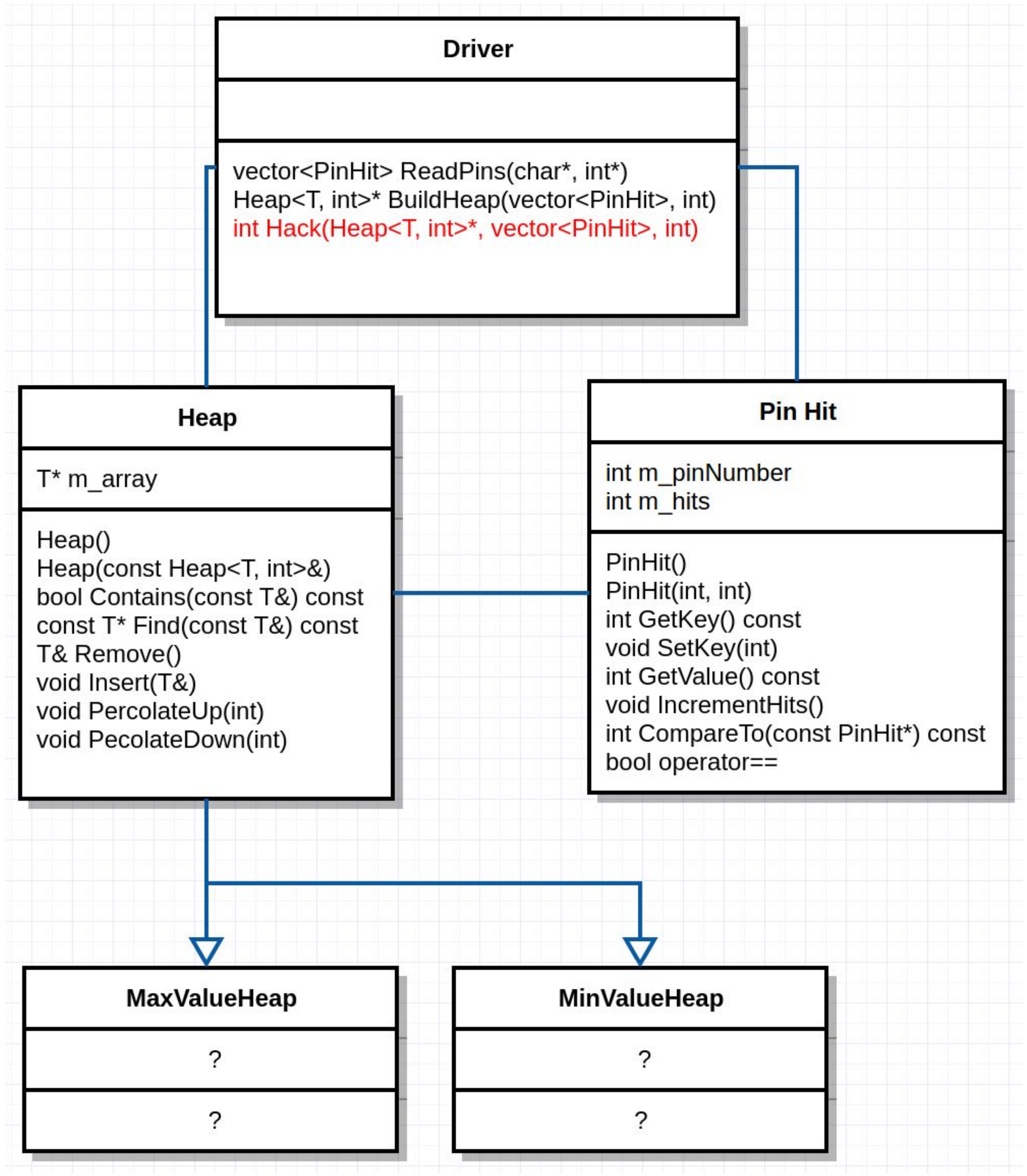
Assuming that Instructor Almes is a real human being, and behaves and acts normally (see: Normality), determine how many of his fictitious pin numbers you can hack before his bank blocks you! His fictional bank with fictional cards and “normal pins” will allow you to incorrectly attempt to access his card 3 times per hour. You have from the start of the assignment to the end to hack as many accounts as possible (14 days * 24 hr/day * 3 attempts/hr = 1008 attempts per card).

You will accomplish this by implementing min and max heaps. You will store pins that you have read from a file into a vector, and increment a hit counter associated to each pin. This vector will then generate the min/max heap. The provided hack method will then generate ‘normal’ pins, test them against a faux bank using your generated heaps and attempt to compromise as many pins as possible before returning a total count of successful hacks. The count is for your prosperity and will only be used for bragging rights. Since your max heap will be prioritizing the most commonly used pins, you should expect better odds than brute force (1:10).

Implementation of a min a heap is required in order to a) demonstrate your understanding of heaps and b) show that ‘secure pins’ are not inherently secure, but more simply, used less in a normal population.

There is nothing inherently secure about a number like 8068 other than the fact that a large majority of people do not use that pin. If you knew someone read this blog, and is a securiphile, you may be inclined to start from the bottom of the list during your brute force attack against them.

UML



* Red: Instructor Supplied

Operations:

The following sections describe how your program should be implemented.

Heap

Implement a heap using templates that take in a class ‘Type’ and an int HeapSize. The heap should be a single dynamically allocated array of ‘types’. Creating a heap could look like: ‘Heap<SomeClass, 32>();’ You may promote UML items in private to protected for this project if you choose to override them.

Heap()

The default constructor for a Heap. Initialize it however you see fit.

Heap(const Heap<T, int>& origHeap)

This constructor copies all the data members from the original heap and saves them in a new heap.

bool Contains(const T& needle)

The contains method returns true if the needle is found in the Heap.

const T* Find(const T& needle) const

The contains method returns an object pointer of type T if the needle is in the Heap. This function may take up to O(n). (Hint: this uses PinHits operator==)

T& Remove()

Removes and returns a min/max value T (by reference) from the heap. In the process of removal, the heap is updated to maintain heap order. This function should run in O(log n).

void Insert(T& insertable)

Given a T, insert will insert the new object into the heap. If needed, the object will percolate up. This function should run in O(n).

void PercolateUp(int index)

This method is used internally on insert. This function should run in O(log n)

void PercolateDown(int index)

This method is used internally on insert. This function should run in O(log n)

MinValueHeap

A min count heap is a heap that uses the smallest values of a comparable (< == >) to prioritize items. For example, the root node will always have the smallest value. In this case, it will be the PinHit with the least hits. This class inherits from Heap. The implementation for this class is up to you.

This **MUST** have a copy constructor that takes a const Heap<T, m_size>& which will call the Heap's (base class) copy constructor (you can put this entirely in the header, it's only 1 line). This is needed for Hack.

MaxValueHeap

A max count heap is a heap that uses the largest values of a comparable (< == >) to prioritize items. For example, the root node will always have the largest value. In this case, it will be the PinHit with the most hits. This class inherits from Heap.The implementation for this class is up to you.

This **MUST** have a copy constructor that takes a const Heap<T, m_size>& which will call the Heap's (base class) copy constructor (you can put this entirely in the header, it's only 1 line). This is needed for Hack.

PinHit

Pin hits are simply Key Value pairs where the Key is the pin number and the Value is the number of times that pin number was seen in the dataset. It's constructor will take an integer pin and optional number of hits with a default of 0 hits. Be sure to check that pins are between 0000 and 9999.

PinHit()

Initializes a PinHit with default values for PIN and frequency. By default the pin number and number of hits should be -1,0.

PinHit(int key, int value)

Initializes a PinHit with the PIN and the number of hits (occurrences) it has.

void SetKey(int pin)

Sets the value of the pin.

int GetKey() const

Returns the pin number, used as a key in this key-value pair.

int GetValue() const

Returns the number of hits, used as a value in this key-value pair.

void IncrementHits()

Increments number of hits this pin has encountered.

int CompareTo(const PinHit& other)

Compares the Value (hits) of this PinHit. Returns 0 if they are the same, negative if this is less than other or positive if this is greater than other. Note, this does not compare the pin numbers in any way.

bool operator==(const PinHit& other)

Compare the Key (pins) against other for equality. If this pin is equivalent to other's pin, return true. Otherwise, return false.

Driver:

This cpp file is provided. However some methods need to be implemented.

vector<PinHit> ReadPins(char* fileName, int* totalHits)

Read the pin data dump into a vector of Pin Hits. Total hits is an out variable that you will set in the method, tallying the total number of lines (pins) in the file. While "totalHits" is originally passed in as zero, this function will return the final count of totalHits that the vector contains.

After this function runs, the vector could look like:

	[0]	[1]	...	[1001]	...	[2343]	...	[8783]	...
PinNumber	0000	0001	...	1001	...	2343	...	8783	...
PinFreq	234	344	...	1287	...	2763	...	6523	...

Heap<T, int>* BuildHeap(vector<PinHit> PinHits, int slots)

This function takes the vector full of PinHits and the total size of the heap. This also uses the type of heap (--min or --max) from the command line. You may use the global variable heapType provided. (Hint: Pins range from 0 to 9999). It returns a pointer to the heap you dynamically created.

int Hack(Heap<T, int>* heap, vector<PinHit> PinHits, int totalHits)

This function is **provided to you**. It will take in a min or max heap and the vector of pins that you read from the file. It will then randomly pick a pin from that vector and use your heap to try to guess what the pin is. You have 1008 attempts (see above). Upon success, a count of successful ‘hacks’ is returned. Try calling this function and comparing the results of using a min heap vs a max heap.

Input File Example

SomeMadeUpInput.txt

```
0000
1021
0000
9284
0000
2954
1234
3656
4444
3333
0000
5454
9854
1093
2789
3421
8734
8732
1111
0000
3434
0000
6789
6968
9898
1254
1234
0000
…
1114
```

Running and Compiling Requirements

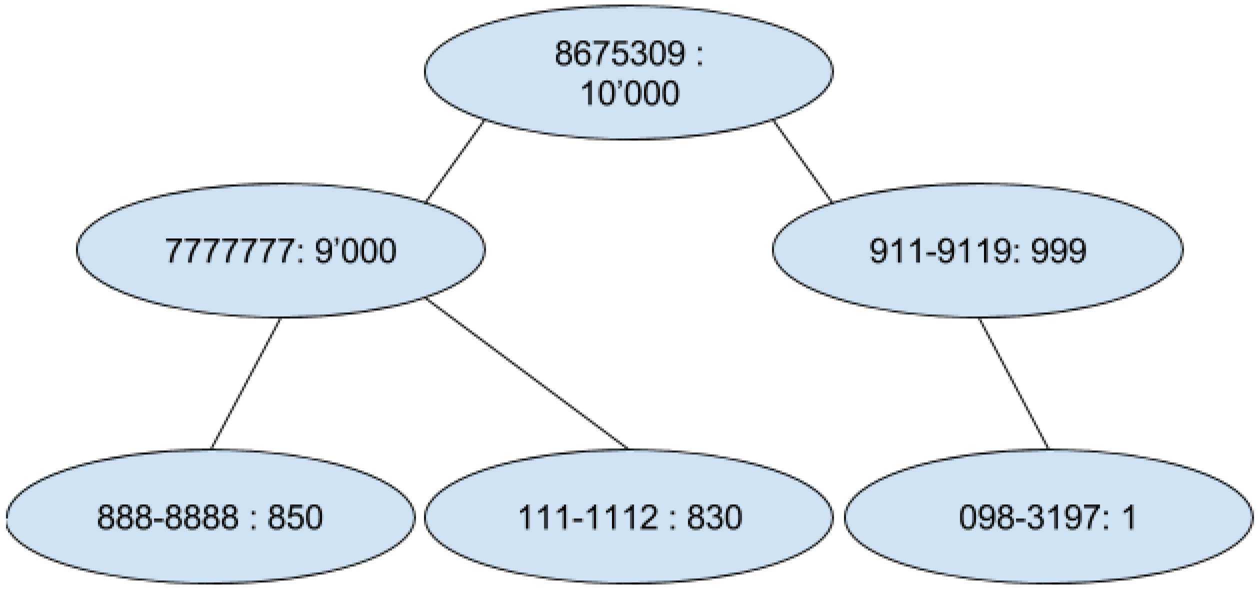
Running the driver, given a list of pins Input1.txt' and the flag for min heaps:

```
./driver.out Input1.txt --min
```

Please compile and complete your work on your own directory on GL. I have had several students that are compiling on mine (or slupoli/pub/cs341...). This takes up a lot of space and my disk quota. My directory is only to be used a repository for your completed files. If this is abused, your directory will be closed automatically by GL

An Example

Below is an example of a max heap that prioritizes popular phone numbers. This heap node is a key value pair of <Phone Numbers, hits> much like your pin hit class:



Fun fact: Cpp 14 allows you to use apostrophes to separate digits in numbers, making it easier to read large numbers at a glance:

- `1234567890`
- `1’234’567’890`

One day in the distant future, when GL is updated, students will be able to use this feature. You don’t have to wait forever though, try it at home or work.

What to Submit

Read the course project submission procedures. *Submission closes by script immediately after 9pm*. Submit well before the 8:59pm deadline, because 9:00:01 might already be late (the script takes only a few seconds to run). You should copy over all of your code under the src directory. You must also supply a Makefile. Do NOT submit your own test data files. **Any unnecessary files submitted will be considered for a deduction**.

Make sure that your code is in the ~/cs341proj/proj4/src directory and not in any other subdirectory of ~/cs341proj/proj4/. In particular, the following Unix commands should work.

```
cd ~/cs341proj/proj4/src

make

make run INPUT=<name of file> FLAG=--<max/min>
ie. make run INPUT=Input1.txt FLAG=--max

./driver.out input.txt --min
(recommended for testing since it's shorter to type)

make clean
```

The command "make run" should simply run the project that compiled successfully. Don't forget the Project Submission requirements shown online!! One hint, after you submit, if you type: ls ~/cs341proj/proj4/ and you see a bunch of .cpp and .h files, this is WRONG. You should see: src instead. The C++ programs must be in directories under the src directory. Your submissions will be compiled by a script. The script will go to your proj4 directory and run your makefile. This is required. You will be severely penalized if you do not follow the submission instructions.

Testing Your Code

Not sure if your heap works? Simply create a test method that removes from your heap and compare them to these pin lists using Input1.txt:

Common Pins (MaxValueHeap)	Secure Pins (MinValueHeap)
1234	8068
1111	8093
0000	6835
1212	7637
7777	9629
1004	0738
2000	8398
4444	6793
2222	8957
6969	9480
9999	0859
3333	6827
5555	7394
6666	6093
1122	7063
1313	8196
8888	0439
4321	8438
2001	9047
1010	9539
0909	6806
2580	7993
0007	8146
1818	8343
1230	8557
1984	3794
1986	6082
0070	7065
1985	8397
1231	8539
1987	8646
1999	9695
2468	9805
1000	0675

Notes

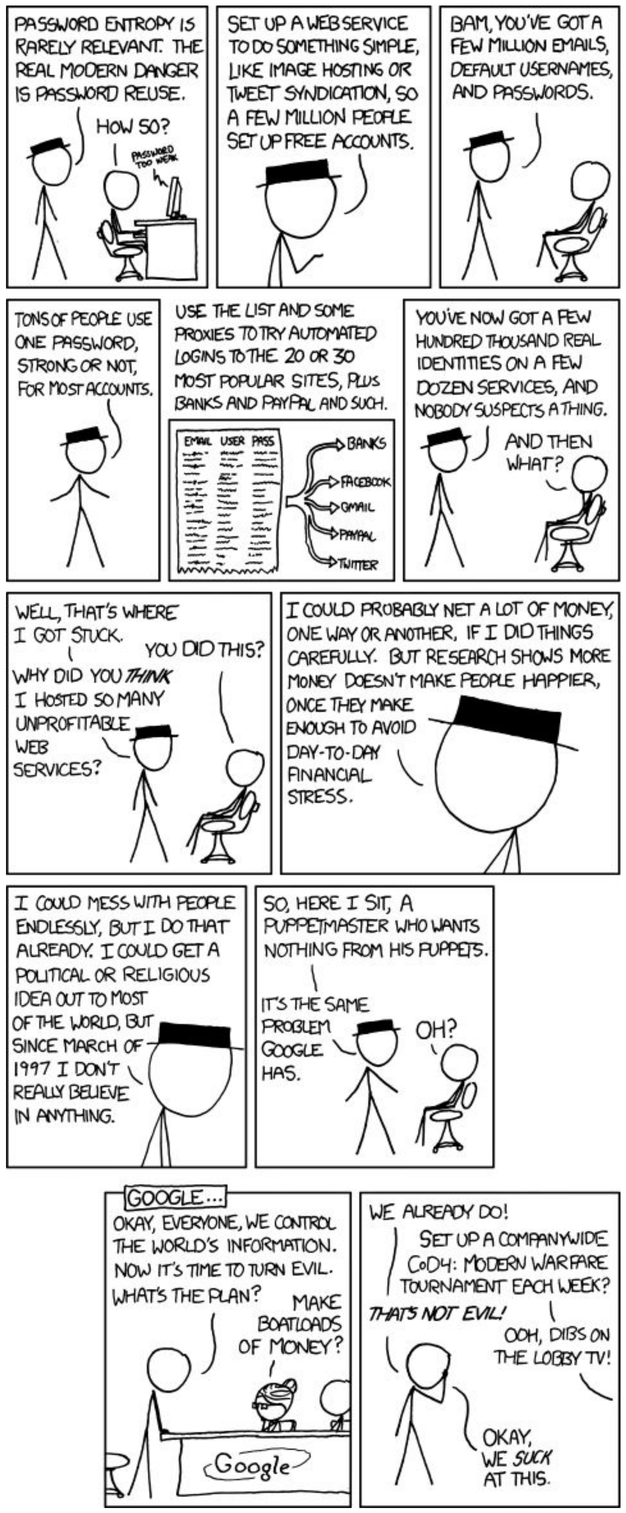
You may have noticed that using a min heap with the BuildHeap method is terribly inefficient (compared to max heaps), as the most common inserts/updates are statistically happening at the bottom of the heap, whereas the max heap was much more frequently updating near the top of the heap. Data structures can be pretty powerful, especially when used correctly.

Also, please note: this project is not going to help you hack bank accounts, or anything for that matter. Modern systems would easily detect this kind of probing for account information. While not nearly as inefficient as brute forcing, a lot of data does depend on more information about the user. IE birthdays, anniversaries, etc. That being said, rainbow tables are a thing.

Please take time to read through the reference material as the article is interesting. And remember, all passwords are very serious and should be created with little to no personally identifiable information. When the majority of the population makes it too easy, who do you think hackers will target first?

References

http://www.datagenetics.com/blog/september32012/
http://kestas.kuliukas.com/RainbowTables/



PASSWORD ENTROPY IS RARELY RELEVANT. THE REAL MODERN DANGER IS PASSWORD REUSE.
HOW SO?
PASSWORD TOO WEAK
SET UP A WEB SERVICE TO DO SOMETHING SIMPLE, LIKE IMAGE HOSTING OR TWEET SYNDICATION, SO A FEW MILLION PEOPLE SET UP FREE ACCOUNTS.
BAM, YOU'VE GOT A FEW MILLION EMAILS, DEFAULT USERNAMES, AND PASSWORDS.
TONS OF PEOPLE USE ONE PASSWORD, STRONG OR NOT, FOR MOST ACCOUNTS.
USE THE LIST AND SOME PROXIES TO TRY AUTOMATED LOGINS TO THE 20 OR 30 MOST POPULAR SITES, PLUS BANKS AND PAYPAL AND SUCH.
EMAIL USER PASS
BANKS
FACEBOOK
GMAIL
PAYPAL
TWITTER
YOU'VE NOW GOT A FEW HUNDRED THOUSAND REAL IDENTITIES ON A FEW DOZEN SERVICES, AND NOBODY SUSPECTS A THING.
AND THEN WHAT?
WELL, THAT'S WHERE I GOT STUCK.
YOU DID THIS?
WHY DID YOU *THINK* I HOSTED SO MANY UNPROFITABLE WEB SERVICES?
I COULD PROBABLY NET A LOT OF MONEY, ONE WAY OR ANOTHER, IF I DID THINGS CAREFULLY. BUT RESEARCH SHOWS MORE MONEY DOESN'T MAKE PEOPLE HAPPIER, ONCE THEY MAKE ENOUGH TO AVOID DAY-TO-DAY FINANCIAL STRESS.
I COULD MESS WITH PEOPLE ENDLESSLY, BUT I DO THAT ALREADY. I COULD GET A POLITICAL OR RELIGIOUS IDEA OUT TO MOST OF THE WORLD, BUT SINCE MARCH OF 1997 I DON'T REALLY BELIEVE IN ANYTHING.
SO, HERE I SIT, A PUPPETMASTER WHO WANTS NOTHING FROM HIS PUPPETS.
IT'S THE SAME PROBLEM GOOGLE HAS.
OH?
GOOGLE...
OKAY, EVERYONE, WE CONTROL THE WORLD'S INFORMATION. NOW IT'S TIME TO TURN EVIL. WHAT'S THE PLAN?
MAKE BOATLOADS OF MONEY?
Google
WE ALREADY DO!
SET UP A COMPANYWIDE CoD4: MODERN WARFARE TOURNAMENT EACH WEEK?
THAT'S NOT EVIL!
OOH, DIBS ON THE LOBBY TV!
OKAY, WE *SUCK* AT THIS.