Name: Jon Abrahamson
ID: 107084898

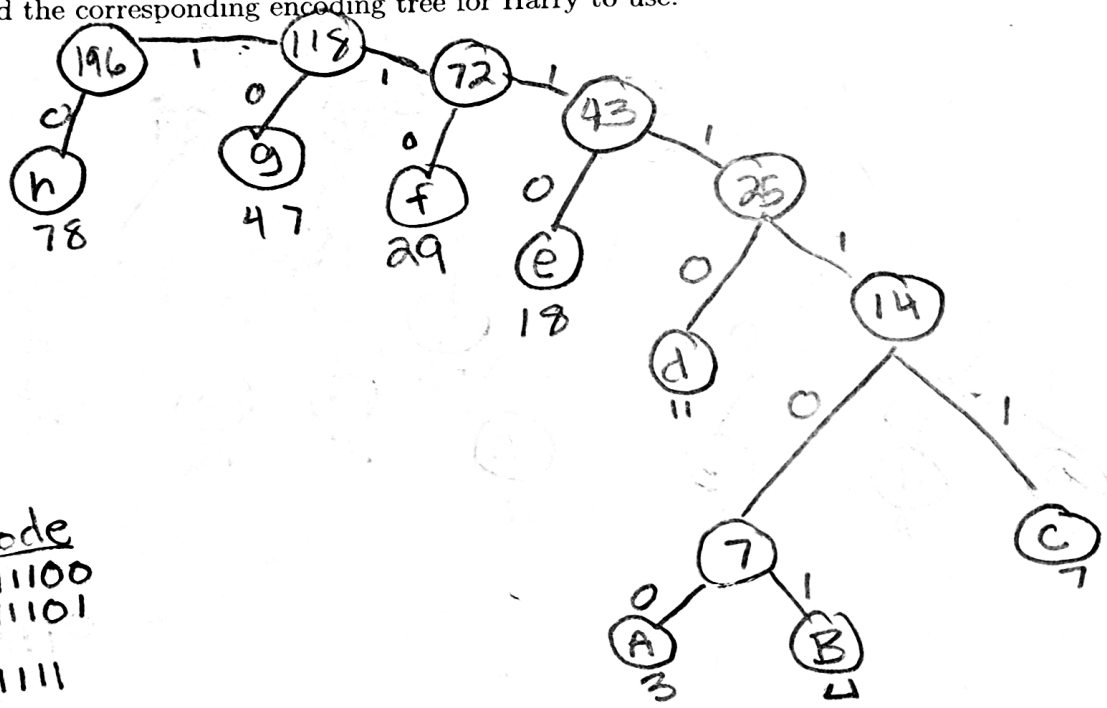
CSCI 3104 **Profs. Grochow & Layer**
Problem Set 5 **Spring 2019, CU-Boulder**

Quick links: 1a 1b 2a 2b 2c 3a 3b

1. Harry Potter is writing a secret message to Hermione and wants to prevent it from being understood by Voldemort. He decides to use Huffman encoding to encode the message. Magically, the symbol frequencies of the message are given by *generalized Fibonacci numbers*, a famous sequence of integers known since antiquity. The nth generalized Fibonacci number is defined by $F_n = F_{n-1} + F_{n-2}$ for $n > 1$ with base cases $F_0 = 3$ and $F_1 = 4$.

   (a) (5 pts) For an alphabet of $\Sigma = \{a, b, c, d, e, f, g, h\}$ with frequencies given by the first $|\Sigma|$ non-zero generalized Fibonacci numbers, give an optimal Huffman code and the corresponding encoding tree for Harry to use.

Letter	Freq
a	3
b	4
c	7
d	11
e	18
f	29
g	47
h	78



Char	Code
A	111100
B	111101
C	11111
D	1111 0
E	1110
F	110
G	10
H	0

Name: Jon Abrahamson
ID: 107084898

CSCI 3104
Problem Set 5

Profs. Grochow & Layer
Spring 2019, CU-Boulder

(b) (15 pts) Generalize your answer to (1a) and give the structure of an optimal code when there are n letters, with the frequencies being the first n non-zero generalized Fibonacci numbers. Prove your answer is correct.

$$L(T) = \sum_{i \in \Sigma} \text{freq}_i \cdot [\text{depth of leaf } i \text{ in } T]$$

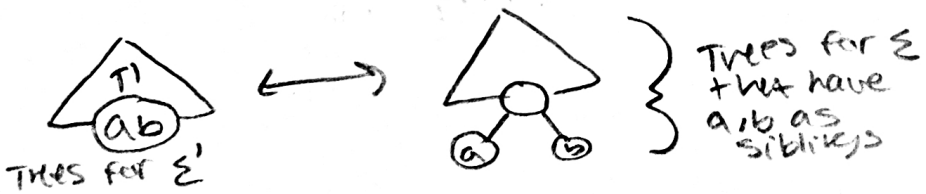
Proof by induction on $n = |\Sigma|$, $n \geq 2$

Base Case: When $n=2$, algorithm outputs optimum tree

Inductive Step: $n = |\Sigma| > 2$.

Inductive hypothesis: if we prove that it is optimal for $n = |\Sigma|$ then for any value less than Σ, it is optimal

Let $\Sigma' = \Sigma$ with a, b replaced by (ab) $\quad \text{Freq}_{ab} = \text{Freq}_a + \text{Freq}_b$



For every pair T' and T, $L(T) - L(T')$ is

$\text{Freq}_a \cdot [\text{depth of } a \text{ in } T]$ + $\text{Freq}_b \cdot [\text{depth of } b \text{ in } T]$ − $\text{Freq}_{ab} \cdot [\text{depth of } ab \text{ in } T']$

1 deeper than ab; 1 deeper than ab

$\text{Freq}_a(d+1) + \text{Freq}_b(d+1) - (\text{Freq}_a + \text{Freq}_b)d = \boxed{\text{Freq}_a + \text{Freq}_b \text{ Independent of } T \text{ or } T'}$ ✓

Name: Jon Abrahamson
ID: 107084895

CSCI 3104
Problem Set 5

Profs. Grochow & Layer
Spring 2019, CU-Boulder

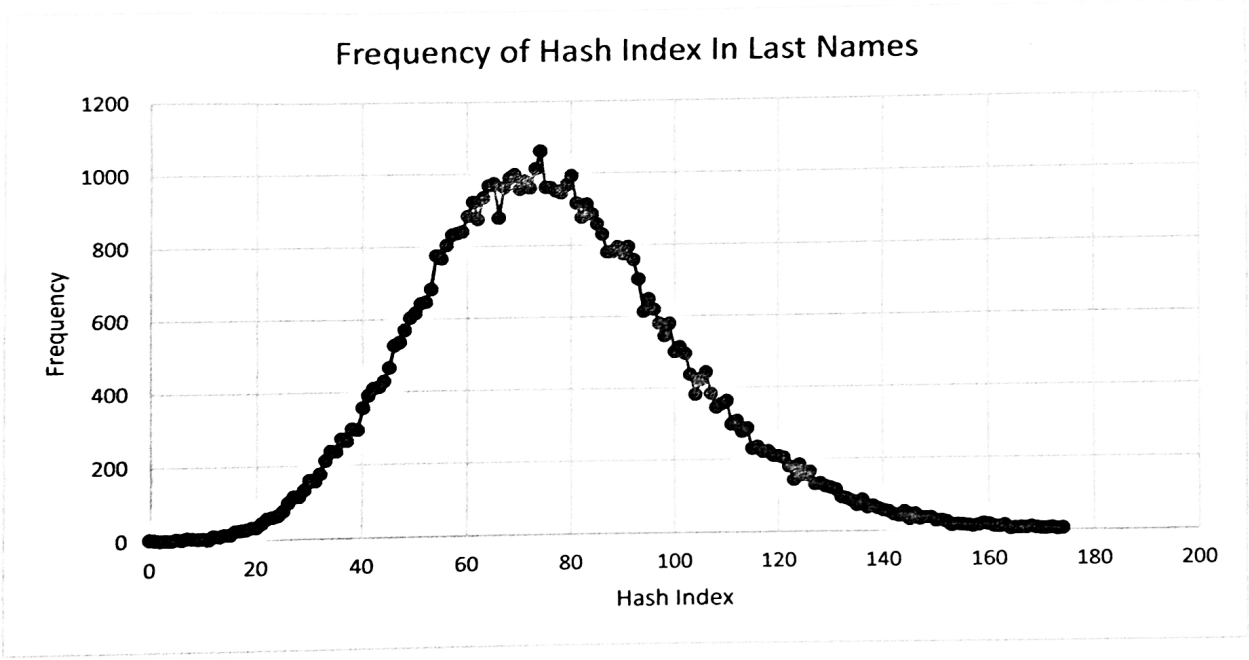
2. A good hash function $h(x)$ behaves in practice very close to the uniform hashing assumption analyzed in class, but is a deterministic function. That is, $h(x) = k$ each time x is used as an argument to $h()$. Designing good hash functions is hard, and a bad hash function can cause a hash table to quickly exit the sparse loading regime by overloading some buckets and under-loading others. Good hash functions often rely on beautiful and complicated insights from number theory, and have deep connections to pseudorandom number generators and cryptographic functions. In practice, most hash functions are moderate to poor approximations of uniform hashing.

   Consider the following hash function. Let U be the universe of strings composed of the characters from the alphabet $\Sigma = [\texttt{A}, \ldots, \texttt{Z}]$, and let the function $f(x_i)$ return the index of a letter $x_i \in \Sigma$, e.g., $f(\texttt{A}) = 1$ and $f(\texttt{Z}) = 26$. Finally, for an m-character string $x \in \Sigma^m$, define $h(x) = ([\sum_{i=1}^{m} f(x_i)] \bmod \ell)$, where ℓ is the number of buckets in the hash table. That is, our hash function sums up the index values of the characters of a string x and maps that value onto one of the ℓ buckets.

   (a) (10 pts) The following list contains US Census derived last names:
   `http://www2.census.gov/topics/genealogy/2000surnames/names.zip`
   (We have also provided a copy of a the CSV file from this ZIP file in the assignment on Moodle.)
   Using these names as input strings, first choose a uniformly random 50% of these name strings and then hash them using $h(x)$.

   Produce a histogram showing the corresponding distribution of hash locations when $\ell = 175$. Label the axes of your figure. Briefly describe what the figure shows about $h(x)$, and justify your results in terms of the behavior of $h(x)$. Do not forget to submit your code with your PS, using the same filename convention as for problem sets `Lastname-Firstname-MMDD-PS5-code.*` (where the "*" can be any plaintext file type you like).

   Hint: the raw file includes information other than name strings, which will need to be removed; and, think about how you can count hash locations without building or using a real hash table.



This shows the collisions that would occur when hashing this amount of data into the limited available bins. It is clear that the collisions follow a standard bell curve distribution centered at a mean of around 75. This shows and is consistent with the fact that a lot of last names are similar in length and have similar characters so there index value calculates to about the same.

Name: Jon Abrahamson
ID: 107084898

CSCI 3104 **Profs. Grochow & Layer**
Problem Set 5 **Spring 2019, CU-Boulder**

THIS PAGE IS A PLACEHOLDER FOR CODE; DO NOT PUT ANYTHING ON THIS PAGE (EVEN CODE) IT'S JUST TO FOOL GRADESCOPE INTO WORKING CORRECTLY

Name: Jon Abrahamson
ID: 107084898

CSCI 3104
Problem Set 5

Profs. Grochow & Layer
Spring 2019, CU-Boulder

(b) (5 pts) Enumerate at least 4 reasons why $h(x)$ is a bad hash function relative to the ideal behavior of uniform hashing.

1) Strings that have the same characters but in different orders would yield the same Hash index, resulting in a collision

2) Given that there are many data points, having only 175 bins does not allow for each string to be assigned a unique index.

3.) Because many names have similar lengths and spellings, we get a lot of collisions around the hash index 75, and not as many on the extremes, it's heavily unequally distributed

4) Because character values are only seperated by a value of "1" you can add an arbitrary value to one letter IE +1 to C to make D and subtract the same value IE -1 from V to make T and this would make strings with different Characters with same Index

(c) (10 pts) Produce a plot showing (i) the length of the longest chain (were we to use chaining for resolving collisions under $h(x)$) as a function of the number n of these strings that we hash into a table with $\ell = 175$ buckets, (ii) the exact upper bound on the depth of a balanced binary tree with n items stored, and (iii) the length of the longest chain were we to use a uniform hash instead of $h(x)$. Include a guide of cn, that is, graph cn on the same chart, for some value of c you find useful, so that you can compare this graph to that of your data.

Then, comment on (i) how much shorter the longest chain would be under a uniform hash than under $h(x)$, and (ii) the value of n at which the balanced binary tree becomes a more efficient data structure than $h(x)$ and separately a uniform hash.

I Don't know

Ran out of time

Name: Jon Abrahamson
ID: 107084898

CSCI 3104
Problem Set 5

Profs. Grochow & Layer
Spring 2019, CU-Boulder

3. Draco Malfoy is struggling with the problem of making change for n cents using the smallest number of coins. Malfoy has coin values of $v_1 > v_2 > \cdots > v_r$ for r coins types, where each coin's value v_i is a positive integer. His goal is to obtain a set of counts $\{d_i\}$, one for each coin type, such that $\sum_{i=1}^{r} d_i = k$ and where k is minimized.

   (a) (7 pts) A greedy algorithm for making change is the **cashier's algorithm**, which all young wizards learn. Malfoy writes the following pseudocode on the whiteboard to illustrate it, where n is the amount of money to make change for and v is a vector of the coin denominations:

```
wizardChange(n,v,r) :
   d[1 .. r] = 1        // initial histogram of coin types in solution
   while n > 0 {
      k = r
      while ( k > 0 and v[k] > n ) { k++ }
      if k==0 { return 'no solution' }
      else { n = n - v[k] }
   }
   return d
```

Hermione scoffs and says Malfoy's code has bugs. Identify the bugs and explain why each would cause the algorithm to fail.

```
WizardChange(n,v,r):
   d[1...r] = 0   #Correction we must initialize with 0 coins each otherwise
                  there would be 1 too many of each coin
   while  n>0 {
        k=r
        while (k>0 and v[k]>n) {k--} # Correction, k goes from
        if k==0 {return 'no solution'}   r to 1 so it must
        else {                           decrease, otherwise
          n=n-v[k]                       we would have an
          d[k]++ }#Correction, we need   infinite loop and
   }                 to update the #     get a run time error
   return d          of coins of each    because k>0 always
                     type in the array d
                     otherwise d would
                     stay at its initialized
                     value
```

Name: Jon Abrahamson
ID: 107084698

CSCI 3104
Problem Set 5

Profs. Grochow & Layer
Spring 2019, CU-Boulder

(b) (8 pts) Sometimes the goblins at Gringotts Wizarding Bank run out of coins,[1] and make change using whatever is left on hand. Identify a set of Euro coin denominations (a subset of the denominations)[2] for which the greedy algorithm does not yield an optimal solution for making change. Justify your answer in terms of optimal substructure and the greedy-choice property. (The set should include the 1 Euro cent so that there is a solution for every value of n.)

Make 35¢ with 25, 15, & 1's.

Greedy: 25, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1

Optimal: 15, 15, 1, 1, 1, 1, 1

The optimal substructure of denominations requires that when a coin is added to the least denomination this value is not less than double the value of the denomination directly less than it.

{1, 2, 3} works because (3+1) ≥ (2+2)

but {1, 15, 25} does not work because (25+1) < (15+15)

[1] It's a little known secret, but goblins like to *eat* the coins. It isn't pretty for the coins, in the end.

[2] `https://www.google.com/search?q=euro+coin+denominations`