*Editors note: A very big thank you to*[*megaterik*](https://www.hackerrank.com/megaterik)*aka Taras Klaskovsky for writing up this editorial, Taras placed 3rd in the 20/20 Hack and graciously shared his code with us so that others can learn.   
  
Please note: Taras skipped, Challenge 2: Click-o-Mania since he used a similar approach with Clobber. And left out Document Classification, however, if you participated in the contest, you can view any of the solutions on the leaderboard. Here is a link to*[*the solutions*](https://www.hackerrank.com/contests/monthly/challenges/document-classification/leaderboard)*.   
  
Also, if you already haven’t signed up, we are having the 101 Hack this Saturday, May 11th from*[*9:30AM to 12:30PM PDT*](http://www.timeanddate.com/worldclock/fixedtime.html?msg=HackerRank%3A+101+Hack&iso=20130511T0930&p1=1241&ah=2)*, we will be giving out 101 prizes for various achievements for hitting the 100k user mark!*[*Click here to sign up!*](https://www.hackerrank.com/101hack?utm_source=mega&utm_medium=blog&utm_campaign=101hack)

**Challenge 1: Ice Cream Parlor**

For each price i we need to check whether prices contains (C - i). This could be done in a variety of ways, for example by adding array of prices to hash map or by using the fact that for larger i (C - i) gets smaller, so that we can use following approach:

int j = n - 1;

for (int i = 0; i < n; i++)

while (price[i] + price[j] > C && j >= 0)

j--;

if (price[i] + price[j] == C)

//Success

**Challenge 3: Alien languages**

First of all, let's notice that word could be divided into smaller "subwords" -- valid words that start with any character and end with character that may be the last letter of a word(i \* 2 > n). We can show that subwords in a word are completely independent -- changing one does not require to change the other. Maximal length of such subword is log(N) (maximal subword consists of characters 1 2 4 8 16 32 ..)

To solve this problem we will use Dynamic Programming approach.

In the start, we compute f[i][j] -- number of subwords of length i, ending with the character j. If character ends with j, it means that previous character was 1, 2, .., or j / 2, or i = 1(first letter could be any) 

f[1][j] = 1

f[i][j] = sum(f[i - 1][k], k = 1...floor(j/2))

Note: f size is N \* log(N), thus simple summation requires N ^ 2 \* log(N) steps, which is too slow for full score, so I suggest to use fenwick or interval tree(or any similar data structure). Having calculated f, we can find how much subwords of length i does this language have as numberOfWords[i] = sum(f[i][j], j = ceil(n / 2)...n)

And finally, using another dp function, we calculate number of ways to make i-word out of subwords 

result[0] = 1 -- empty word of length 0

result[i] = sum(result[i - j] \* numberOfWords[j], j == 1...log(N))

**Challenge 4: Nice Clique**

We need to do exactly what described in the statement: find number of prime divisors and sum of all positive divisors.

Let's start with 80% solution and find prime divisors using "for" cycle from 2 to sqrt(n) and calculate sum of divisors using canonical number representation:

If x = p1α1 \* p2α2 \* ... \* pnαn, then sum of divisors of x is equal to

(1 + p1 + p12 + .. + p1α1 \* (1 + p2 + ... + p2α2) \* ... \* (1 + pn + pn2 + ... + pnαn)   
  
Proof: every summand is a divisor of x, and every divisor of x equals to some summand in this formula.

for (long j = 2; j \* j <= a; j++)

{

if (a % j == 0)

{

numberOfPrimeDivisors++;

long currentValue = 1;

long currentSum = 1;//(1 + x + x^2 + ..) \* (1 + y + y ^2 + ...) + ...

while (a % j == 0)

{

a /= j;

currentValue \*= j;

currentSum += currentValue;

}

sumOfDivisors = (sumOfDivisors \* currentSum) % 2;

}

}

After we got these numbers, we can specify a class for each number:

1. c00 -- (numberOfPrimeDivisors % 2 == 0) and (sumOfDivisors % 2 == 0)
2. c01 -- (numberOfPrimeDivisors % 2 == 0) and (sumOfDivisors % 2 == 1)
3. c10 -- (numberOfPrimeDivisors % 2 == 1) and (sumOfDivisors % 2 == 0)
4. c11 -- (numberOfPrimeDivisors % 2 == 1) and (sumOfDivisors % 2 == 1)

Answer is the maximal sum of size of the two neigthboring(differing in one digit) classes(i.e. max(c00 + c01, c00 + c10, c11 + c10, c11 + c01), because if we take two random numbers from answer set:

1. If they belong to one class it means that both conditions are met
2. If they belong to "neighboring" class(c00 and c01, c11 and c10, but not, for example, c00and c11) it means that one of these conditions is true
3. If they belong to (c00 and c11) or (c01 and c10) it means neither of these conditions is met and such elements coudn't be in answer set

To speed up this solution and get a full score I have generated list of prime divisors with [Sieve of Eratosthenes/](http://en.wikipedia.org/wiki/Sieve_of_Eratosthenes) up to sqrt(max number)

**Challenge 5: Clobber**

Clobber is a very interesting game due to bad perfomance of often used alpha-beta prunning, because it's hard to find heuristic function to assign value to a game state(difference between number of pieces, which performs pretty well in checkers or even in chess doesn't works here at all, because each turn other player loses one piece).

After several attempts to create such function, I abandoned the alpha-beta prunning approach and tried Monte-Carlo Tree Search(MCTS) with UCT. Advantages of MCTS with UCT:

1. No need for heuristic function.
2. With time solution gets better, but you can stop it at any moment and return your best move.
3. With time different paths get another chance.

MGTS is also known for good results in Go, the game in which alpha-beta approach wasn't able to defeat human players yet.

I recommend <http://mcts.ai/> for more information and code templates.

The only thing left between you and third place at clobber is memory optimization: keeping list of node's children(as it is done in mcts.ai java template) increases memory consumption by the factor of 10(and, therefore, increases time), so this needs to be changed -- my node entry takes only about 50 bytes by storing edges in adjacency lists and passing char[][] board as a parameter, not a field.

Some other interesting approaches can be found in the internet, for example [here](http://www.researchgate.net/publication/221932254_New_Trends_in_Clobber_Programming).

**Challenge 2: Click-o-Mania:**

I used the same approach in Clobber, it wasn't stable on the largest test (75 out of 80 points in average), but thankfully HackerRank doesn't limit number of submits :)

Please discuss on [Hacker News](https://news.ycombinator.com/item?id=5687251).