Jules Voltaire

CS2223 – Algorithms

Project 1 Report

In this project, two algorithms were used to detect anagrams. The first one is an easy and slow algorithm, brute-force. The worst-case for this one is n! which is grows quite quickly but no additional space was necessary. The second one is a count and compare algorithm. Although the time efficiency is O(n) however additional space was required to achieve this algorithm.

Since in an anagram, all the letters in one string have to be present in the other, both strings therefore have to have the same length. In that case, both algorithms check to make sure that the length of both string are the same and immediately return false if they are not.

The brute-force algorithm is very slow as it grows really as the length of the string increases n! to be exact. This is achieved by first generating a list of permutations of one of the strings, then iterate over that list to see if the other string in there. In my case, a generator object is returned instead of a list and a for each loop is used to iterated over that generator object. See below for the full Pseudocode code.

**Brute-Force(str1, str2)**

if(length of str1 is not equal to length of str2)

return false

else

for each item in stringPerms(str2)

if(str1 is equal to current item)

return true

return false

**stringPerms(str)**

if(length of str is <= 1)

yield str

else

for each result stringPerms(str[1 to end])

for i = 1 to length of str

yield all letters in result from 1 to i plus

the first letter from str plus

all letters in result from i to length of result

As a result, brute-force could only be tested with small-medium strings (<13). The graph below shows how fast the graph grows from strings of length 10 to strings of length 12.

Once, asserted that the two strings are of the same length, the count and compare algorithm iterates over each string consecutively counting how many of each letter from the alphabet is present then the result for each string is then compare to see if both strings contains the same number of the same letters. See below for the full pseudocode. This algorithm sacrifices space to make up for time.

**countAndCompare(str1, str2)**

isAnagram = True

if(length of str1 is not equal to length of str2)

isAnagram = False

else

cnt1[26]

cnt2[26]

for(i = 1 to length of str1)

determine what the index for the current letter (str1[i]) should be in cnt1

increment the count for that letter in the cnt1 by 1

for(i = 1 to length of str2)

determine what the index for the current letter (str2[i]) should be in cnt2

increment the count for that letter in the cnt2 by 1

for(j = 1 to 26)

if(cnt1[j] is not equal to cnt2[j])

isAnagram = False

break // we don’t have to keep going

return isAnagram

To conduct this experiment, two very import websites were used. The first one is from [random string generator](http://textmechanic.com/text-tools/randomization-tools/random-string-generator/). This tool was used to generate random strings of a given length. This data is then pasted in the command line for testing. The second useful website is a [permutation generator](http://users.telenet.be/vdmoortel/dirk/Maths/permutations.html). Once a random string was generated, in order to test a positive anagram, this tool was used to generate a list of random string permutations from which one is randomly picked to be tested. One important factor to note for the brute-force algorithm is that the plot points can be a little dispersed for strings that are anagrams because depending on the randomly drawn permutation and where it is located in the list of permutations, it may take longer to find.

To time these algorithms, the current date and time is log, then the anagram detection is performed. Once done, the current date and time is logged again and finally the time before is subtracted from the time after. See table below for all data collected.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| String1 | String2 | Are Anagrams | NumberOfCharacters | Brute Force (s.us) | Count and Compare (s.us) |
| qs | ks | FALSE | 2 | 0 | 0 |
| gj | jg | TRUE | 2 | 0 | 0 |
| bvtb | qlco | FALSE | 4 | 0 | 0 |
| zlyq | zqyl | TRUE | 4 | 0 | 0 |
| bastut | jshrlx | FALSE | 6 | 0.504 | 0 |
| bgbhsb | bhbbsg | TRUE | 6 | 0.503 | 0 |
| biscrmms | ubvqxitl | FALSE | 8 | 0.027012 | 0 |
| kzfuncru | unfkzcru | TRUE | 8 | 0.0005 | 0 |
| kzfuncru | cunzfruk | TRUE | 8 | 0.019539 | 0 |
| ptqlskpcm | qnnzjlugj | FALSE | 9 | 0.237085 | 0 |
| yigkhtjgz | gghizjtyk | TRUE | 9 | 0.108541 | 0 |
| ewriltczvh | wrltcxqspy | FALSE | 10 | 2.336395 | 0 |
| nbrwvtclwr | blrtrwvnwc | TRUE | 10 | 0.277644 | 0 |
| nbrwvtclwr | cvwrwlrntb | TRUE | 10 | 1.403989 | 0 |
| ilnhytyvgof | xmisaperjhj | FALSE | 11 | 25.517634 | 0 |
| tcnjtkhgwis | gckjhsitwnt | TRUE | 11 | 9.621463 | 0 |
| evenqfixzzbj | hibtlnbutrfm | FALSE | 12 | 300.508775 | 0 |
| knszhbfxptfk | nkkhbstpfxzf | TRUE | 12 | 233.310454 | 0.000499 |
| Do an angry hit | Tonya Harding | TRUE | 12 | 106.58294 | 0 |
| Do an angry hit | Tonyo Hardigr | FALSE | 12 | 301.112837 | 0 |
| kctyrtqgounptwyxfxwevahkuhuageojhuxfoemppugozrcmnn | rnjwnagcqwnulfoloegchzmxwflnicljjybmxkzpqfxigiikwj | FALSE | 50 | --- | 0 |
| aawnwhrpyhnauqkvzvweipkikobekqxyuvpnrcasofmazsrlxm | aawnvpwhrpyveipkikobekqxyunrchnauqkasofmazsrlzvwxm | TRUE | 50 | --- | 0.000503 |
| kisenzczpulzthzyfnmhbkisqknszqejmhgcsqnlngztxrpjhqaurcrj esofvrbuhrnubiumyapsajjinrpphgtpitsrtrgelata | ewlimpntljrafmgakwznqibzwrakpvnfupniqkxnbpwkkmqpku rtpeajzetfwrlkhepqhxrcfgesfnilqovnqgepuxljkwcibzoe | FALSE | 100 | --- | 0 |
| ayviyavialiophuraaynvlarngjozbzyywvjumjcnkvznhep bnmllxmezvmjjfpwksagnqiuhzjxpxhroirbknucfccpqgbipvelb icjnyjezswotulabvjrxeksicxqzraijcygjsojanipevctboekwobgivr ugfcbcwbtrgihqhnennshknmjvwyvbfyoqgcylveymojqxjmwhlfi rxynbcishjcgtmimwgcrkaurzkukgljvivslbmqzncyltuwnwvnmlh ulkuqlceoygqkcrsmablrszclohllcqaxggcwsuyyyfvlbrjunsvevbtg lecmwcprijnckryylntbzmqtyezhqmjpuusqesojhqhvvbvorfnjxm ckwxnwprkpzmxzuxmohlcfcrprnbtxcmqupkavbsztgiyvkiqgmj ruqnegbumnfatbjtgyztykaqvaihzqxaenzxaftbsstsfswmqerqe wozrutptmkvzjoeoccalhojetxfwahtxctrfyevhreeqasjsyyhae ragqtqlgfszwmjtlplyjmyogpahnezlzfzlupbzbkeiutvnlctmrgkka tnpswcypzmlfrbrtothfoljtwortsieihklfqhcnosfmokhjrwiaszet xyxnnuiyxxutixoaelkjkieszitunvjfasnhtoeykebpjvsxlxghbmop nzxpozmxyguheaxqbybhxwpnwltauywysqckfryvzmuitgqcfxtg uzjwklfbjxacyjclyighoqjtmlyolwcaargkbxcgpsqxrezfaartmwav jspowlowaihebveyugqesresvbopbbxbnrlwlvtmpxzhrcrpbeqq opfpmaptojhmuowfovrysvnttxmroqnlhnkpqgwkvfvfrxtzttss oelgbvqpeozvxowefcfqnqulvamuwjullqnorybzlaqqgzkcqjiov iccynwctcjsvnegxqanxlkmhslaszryn | kcmcixkoqeruobmmfmxfjqauoznwkqkbatrbxvspkywutcio uzymoqpvktybnamihwrkkimsxmimvxzkhrxllsyxnmxxlpnk eytulhaybtwgqgaouklszutnijshxuhjkgutecmhqiaxjayiibau hmkxyjlebraoaehmtaywiivqyuepelzzgtooogynhwmtsheoar giszqnvkbljljjczuvmuymoimejfpzncwflejithrwxzfjprszzuiiw izrlhpyqxxnicbgetbkmpihmffwkuumpsgruwhlbghrlusaxezw rwciglkujxrbqotygnippbynywgzyqkvqxjrfbtrvypkocpnzlpkp hlqmaljlptonqrtyjafhxwrsntqtlkibvaisblnybpiynweyeevpijfjjp hsuxncvhlwivfewtmqxiihowahacxvtecmfsnzbmvantsyssnxt txvnvvyjolvskqvabnwgzgtmcupjktfzosbjhgrgrhkvmivwootha ospqlyrhgykeoacppajclxoznsccpoywckonxqpylyuoyeesocqklc awbkcymcawtrtpzbvkefqbsnmirxvkfpvrorqskuqpcsowzszrhg hcyvrsaijwomfxnyhkzmqwsaxrcvpvyouqzbarnmnwbrqeylnrr zjosquayhvbsbbikpyxtzebluezjzovhbozxbbqntcqzgnerqmsb txtlllebhkppiziwnhwwuuwwistttfgzcmomapomyehgmfhf pnmysjueaaifkuscllurxjkqrkptnrswtnkhnsrcovyuequuipgzq hgebvhhsfkhmjhlarltqptwswqgcjkoituwuwfojazowjolcbq uhpapcfnauvhfcjqqewwlvgxoaptoyjblbrlytcbccuunqaeine vhtbcwjevjhabpmgizxnuxpvthsgtwtscgbafeoapcqkmooqfnf vojtthwqfetjrlwgccj | FALSE | 1000 | --- | 0.000502 |