Hidden Password

Set up

It was written in **C++ 11**. Run by command

`g++ -lm -lcrypt -O2 -std=c++11 -pipe -DONLINE_JUDGE`

It has been accepted by Uva.

Algorithm overview

1. Grow the string by appending itself
    1. ex. abc => abcabc
2. Build suffix array
    1. Make use the implementation of suffix array in stanford notebook
3. Loop the suffix array from top, `suffix.L` = twice of the original string length
    1. With this method we always get lexigraphically smaller suffix
    2. If the length of the suffix is larger than `suffix.L/2` , in another word, it has a prefix of length L, it is what we are looking for.
    3. But there may be the case that multiple suffixs have same length of L prefix, we need to take the lexigraphically larger one among
        1. ex.

            abcabcabc

            abcabcabcabc

            abcabcabcabcabc
        2. We take the last one since the first "a" has the smallest index in the string

Implementation detail

1. Loop suffix array from top
    1. Build pair and sort according to SA[i]
    2. If case 3.3 from Algorithm overview shows up, skip the iteration.

GATTACA

How to run

The program is written in C++ 11 standard and tested and built under Ubuntu with GCC 5.4. `g++ -std=c++11 -Wall -Wextra dna.cpp`

How does the program work?

- For each case, read in the string and build the suffix array.
- Using the suffix array, we are able to find the Longest Common Prefix for the string for i = 0 to length of the given string.
- Now we already know the longest repeated part of the string. We now need to find how many times this string is repeated.
  - In order to achieve this, I used the built-in function `string::find`. Every time I am able to find the longest repeated string, I will increment a counter.
- After calculated how many times it has been repeated, we are ready to print out the result.

Data Structure

1. Suffix array implementation is modified from Stanford ACM codebook and make use of the Longest Common Prefix function defined in the data structure.
2. A custom sorted set is used to keep track of which is the longest repeated string. This is done by using a lambda function like this:

```
auto sort_set = [](const string& s1, const string& s2) {
    if (s1.length() == s2.length()) {
        return s1 < s2;
    } else {
        return s1.length() > s2.length();
    }
};
```

- Once an element is inserted into the set, we sort it using the length in descending order. If 2 elements in the set have the same length, we will sort alphabetically. This guarentees that the first element in the set is what we want.

Where's Waldorf?

How to run

Tested and built under Ubuntu with GCC 5.4. `g++ -std=c++11 -Wall -Wextra PokerHands.cpp`

How does the program work?

- First, read in all inputs and constuct a data matrix consisting of the puzzle that we need to work on.
- For the words we need to find, we need to concatenate them with the character `.`. For example, if the test words are `Waldorf`, `Bambi`, `Betty`, we would need to turn these words into `.Waldorf.Bambi.Betty.`.
- Next, we would need to build a suffix array based on the resulted string of the previous step. The code for constructing such data structure is adapted from Stanford ACM Codebook.
- On top of that data structure, I've written some code to do **binary seach on the suffix array** to find whether a given string is an exact match in the suffix array. For example, `Betty` is an exact match of the constructed string whereas `Bet` is not. In order to compute this, basically I need to find whether `"." + the_given_string + "."` is in the suffix array or not.
- When the proram runs, it will loop through each position for each row and for each column. During each position, the program will construct 8 strings from that position. They are `horizontal_right`, `horizontal_left`, `vertical_up`, `vertical_down`, `diagonal_right-up`, `diagonal_left-down`, `diagonal_left-up` and `diagonal_right-down`. They represent the 8 directions where a given position can reach. For each direction string, we need to call `is_exact_match()` to find whether such string can be exactly found in the suffix array. If yes, we will record the string that can be found from this location and the coordinate of the location.
- In the end, the program will output the locations that has been recorded previously and from where the given test words can be found.

LifeForms

- How to run
  - In UVa: C++11 5.3.0 - GNU C++Compiler with options: -lm -lcrypt -O2 -std=c++11 -pipe -DONLINE_JUDGE
  - Local machine: g++ -std=c++11SuffixArray.cpp
- Idea
  - In one scenario, connect all formsinto one string/char array
    - Separate these forms by usingcharacter that is not appeared in the forms
    - E.g. “abcdefg” and “bcdefgh” ⇒abcdefg.bcdefgh
  - Use Suffix Array algorithm to getsuffix array (sa)

- Get height array by using suffixarray, according to The Kasai et al. algorithm (http://www2.cs.sfu.ca/~binay/2018/409/suffix-array.pdf)
  - Use binary search to get the rightlongest common substring length
    - The condition within binary searchis that if the height array has the height that appeared one than half of n,then it is valid. Otherwise, it is invalid.
  - Print results by requirements.
- Data Structure and Algorithm
  - Suffix array
    - Modified from https://github.com/jaehyunp/stanfordacm/blob/master/code/SuffixArray.cc
  - The Kasai et al. algorithm
    - https://www.hackerrank.com/topics/lcp-array
    - https://pdfs.semanticscholar.org/f5e5/f365acc6f00c014c523c65efd9df6cee2606.pdf
    - http://www2.cs.sfu.ca/~binay/2018/409/suffix-array.pdf
  - Binary search