CS2010 PS1 - Baby Names

Released: Thursday, 25 August 2011
Due: Saturday, 03 September 2011, 8am

Collaboration Policy. You are encouraged to work with other students on solving this problem set. However, you **must** write up your solution **by yourself**. In addition, when you write up your solution, you **must** list the names of every collaborator, that is, every other person that you talked to about the problem (even if you only discussed it briefly). Any deviation from this policy will be considered cheating, and will be punished severely, including referral to the NUS Board of Discipline. It is not worth it to cheat just to get 15% when you will lose out in the other 85%.

The Story. This is the first problem set for CS2010. As you may have known, your lecturer Steven is going to be a father soon (expected around mid October 2011 – that is, in the middle of this semester[1]. So, Steven has decided to give a grand theme to CS2010 this semester: 'Babies'. Many, if not all of you, will probably encounter these series of 'real life problems' if you have wife/husband and bab(ies) in a few years time. Steven hopes that CS2010 students find this semester a bit more special by knowing that all these 'real life problems' can be 'solved' with what you learn in CS2010. Note: PS1-8 are chronological.

The first PS1 happened when Steven and his wife (Grace Suryani) discovered that Grace is pregnant (this was back in mid February 2011). After confirming the result of our home pregnancy test kit (the 'two stripes' == pregnant) with a gynaecologist, we started to break this news to our parents, to my brother, then to our friends. And soon... we encountered this situation: My parents gave a few name suggestions, my wife parents did the same thing, and so did some of our friends. We ourself have a few ideas for our baby names.

In the past (when Steven was a baby himself), nobody knows the gender of a baby until he/she is born. Today's medical technology (ultrasound scan) can detect the gender of the baby roughly around the second trimester of pregnancy. Thus, at the early stage of pregnancy when we still did not know the gender of our baby, we received suggestions for both male and female baby names.

Eventually, we have to select one name. Steven, being a computer scientist, wants to use his Computer Science knowledge to help him answering some queries.

Steven and his wife, after knowing that the gender of our baby is female (at around June 2011), eventually decide that her name will be *"Jane Angelina Halim"*. "Jane" means "God is Gracious", "Angelina" means "God's messenger", and "Halim" is Steven's family name.

[1]Btw, Steven is still in process to find replacement lecturer(s) in those affected week(s).

The Actual Problem. Given N baby name suggestions (each baby name consists of *only* uppercase alphabet characters of no more than 1500 characters) and the gender suitability of that name (integer 1 for male or integer 2 for female), tell Steven how many baby names start with a *prefix*[2] that is inside a given query interval `[START..END)`, where[3] `START` < `END`, and both are strings. Notice that the interval is left-closed and right-open. There are Q queries that you have to answer. Use the most efficient technique that you learn so far.

The skeleton program `BabyNames.java` is already written for you. You just need to implement mainly these three more methods/functions:

- `void AddSuggestion(String babyName, int genderSuitability)`
  Insert `babyName` and its `genderSuitability` into a data structure of your choice.
- `int Query(String START, String END, int genderPreference)`
  Query your data structure and report the number of baby names that start with a prefix that is inside the query interval `[START..END)`.
  if `genderPreference = 0`, report the number of both male and female baby names.
  if `genderPreference = 1`, report the number of male baby names only.
  if `genderPreference = 2`, report the number of female baby names only.
- If needed, update the constructor of Class `BabyNames`.

Example:
Let there be $N = 4$ baby names: {(ROBERT, 1), (JANE, 2), (MARIA, 2), (PETER, 1)}.

- `Query(''PET'', ''STE'', 1)` = 2 as we have (ROBERT, 1) and (PETER, 1).
- `Query(''PET'', ''STE'', 2)` = 0 because although we have ROBERT and PETER, both are *not* female baby names.
- `Query(''JA'', ''PETA'', 0)` = 2 as we have (JANE, 2), (MARIA, 2). Notice that PETER is *outside* the query interval `['JA'..'PETA')`.
- `Query(''PET'', ''ROB'', 1)` = 1 because ROBERT is outside the query interval `['PET'..'ROB')`. Remember that the interval is left-closed and right-open.
- `Query(''JANE'', ''MARIA'', 2)` = 1 because JANE is a female baby name that has prefix inside the query interval `['JANE'..'MARIA')`, but MARIA is not.
- `Query(''JANE'', ''MARIANA'', 2)` = 2, now JANE and MARIA are inside the query interval `['JANE'..'MARIANA')`.

Subtask 1 (40 points). $Q \leq 10$, $N \leq 26$. All baby names have distinct first letter. All queries have `END` equal to one character after `START`; the maximum `END` is 'Z'; both `START` and `END` only contains 1 character; and `genderPreference` $\neq$ `0`.

Subtask 2 (30 points). $Q \leq 10$, $N \leq 26$. All baby names have distinct first letter. All queries have `START` < `END`; both `START` and `END` only contains 1 character; and `genderPreference` can be {0, 1, 2}.

Subtask 3 (20 points). $Q \leq 10000$, $N \leq 10000$. With $N \leq 10000$ (yes, there are lots of name suggestions for Steven and Grace), there will obviously be several names with the same first letter. All queries have `START` < `END`; `START` and `END` can have more than one characters; the gap between `START` and `END` are designed to be small (e.g. `[''SA''..''STR'')`, `[''PE''..''PO'')`, etc), and `genderPreference` can be {0, 1, 2}.

[2] A prefix of a string $T = T_0T_1...T_n$ is string $P = T_0T_1...T_m$ where $m \leq n$.
[3] In Java, you can compare two strings using `compareTo` method.

Subtask 4 (10 points). Everything else similar to Subtask 3, but the gap between `START` and `END` can be the *maximum* possible (e.g. `[‘‘A’’..‘‘ZZ’’)`, `[‘‘AB’’..‘‘YZ’’)`, etc).

Bonus point: 10 points. If you can solve Subtask 4 with *your own* Binary Search Tree (that is, your solution code cannot use Java TreeMap or TreeSet at all!), you will get 110 points.

Note 1: The test data to reach 90 points: `Subtask1.txt`, `Subtask2.txt`, `Subtask3.txt` are given to you. You are allowed to check your program's output with your friend's.

Note 2: If you encounter stack overflow, try running your Java program with: '-Xss8m' flag. Ask your Lab TA if you are not sure about the meaning of this flag.