CS2010 PS1 - Baby Names v2 (with R-option)

Released: Tuesday, 28 August 2012
Due: Saturday, 08 September 2012, 8am

Collaboration Policy. You are encouraged to work with other students or teaching staffs (inside or outside this module) on solving this problem set. However, you **must** write the Java code **by yourself**. In addition, when you write your Java code, 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). This list may include certain posts from fellow students in CS2010 IVLE discussion forum. 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%.

R-option. This PS has R-option at the back. This additional task usually requires understanding of data structure or algorithm *beyond* CS2010. A self research on those relevant additional topics will be needed but some pointers will be given. CS2010R students *have to* attempt this R-option. CS2010 students can choose to attempt this R-option too for higher points, or simply leave it.

Background. This is the first problem set for CS2010/R. As you may have known, your lecturer Steven is already a father since last year (to be precise, since the arrival of his first baby on 24 October 2011). Last year, Steven has decided to give a grand theme to his CS2010: '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. Since this 'value added feature' of CS2010 is well received among your seniors last year, Steven has decided to reuse the same storyline, although now they are all past events (until Steven has his second baby). 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: PS1-7 are chronological. PSBonus and PS8 are not part of this storyline.

Last Year's Story. The first PS1 happened when Steven and his wife (Grace Suryani) discovered that Grace was 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 baby name suggestions, my wife parents did the same thing, and so did some of our friends. We ourself had a few ideas for our baby's name.

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 had to select one name. Steven, being a computer scientist, wanted 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 decided 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.

The Actual Problem. Given N *distinct* baby name suggestions (each baby name consists of *only* uppercase alphabet characters of no more than $\mathbf{M = 15}$ **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*[1] that is inside a given query interval `[START..END)`, where[2] `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 have learned 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)`, depending on parameter `genderPreference`:
  - 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 $N = 4$ distinct 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')` as 'PETER' ≥ 'PETA'.
- `Query(''PET'', ''ROB'', 1)` = 1 as we have (PETER, 1). See that ROBERT is outside the query interval `['PET'..'ROB')` as 'ROBERT' ≥ '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 as 'MARIA' ≥ 'MARIA' (actually they are equal). Remember that the interval is right-open!
- `Query(''JANE'', ''MARIANA'', 2)` = 2, now JANE and MARIA are inside the query interval `['JANE'..'MARIANA')` as 'MARIA' < 'MARIANA'.

Subtask 1 (25 points). $Q \leq 10$, $N \leq 26$. All baby names have distinct first letter. both `START` and `END` only contains 1 character (the maximum `END` is therefore 'Z').

Subtask 2 (Additional 25 points). $Q \leq 10000$, $N \leq 10000$. With $N \leq 10000$ (yes, there are lots of different name suggestions for Steven and Grace), there will obviously be several names with the same first letter. Now, `START` and `END` can have more than one characters but the gap between `START` and `END` are designed to be small (e.g. `[''SA''..''STR'')`, `[''PE''..''PO'')`, etc).

Subtask 3 (Additional 25 points). Everything else similar to Subtask 2, but the gap between `START` and `END` can be the *maximum* possible (e.g. `[''A''..''ZZ'')`, `[''AB''..''YZ'')`, etc).

[1]A prefix of a string $T = T_0T_1...T_{n-1}$ is string $P = T_0T_1...T_{m-1}$ where $m \leq n$.
[2]In Java, you can compare two strings using `compareTo` method.

Subtask 4 (Additional 25 points). To get 100 points in this PS, you have to solve Subtask 3 above with *your own* balanced Binary Search Tree (that is, your solution code cannot use Java TreeMap or TreeSet at all!).

Note 1: The test data to reach 50 points: `Subtask1.txt` and `Subtask2.txt` are given to you. You are allowed to check your program's output with your friend's. You are encouraged to generate and post additional test data in IVLE discussion forum.

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.

R-option (110 points). Given N *distinct* baby name suggestions (each baby name consists of *only* uppercase alphabet characters of no more than M characters), tell Steven how many baby names **contain *substring*[3] that match query substring** `SUBSTR`. There are Q queries that you have to answer. Note that for this R-option, we still use the same constraint size: $Q \leq 10000$, $N \leq 10000$, but this time baby name can go up to $M = 100$ characters. However, we guarantee that the answer for each query will not be larger than 100.

The skeleton program `BabyNamesR.java` is given. The important modifications are:

- `void AddSuggestion(String babyName)`
  We drop `genderSuitability`.
- `int Query(String SUBSTR)`
  We change `START` and `END` to just one parameter: `SUBSTR` and drop `genderPreference`.
  We guarantee that this function will never report answer larger than 100.

Using these 5 *distinct* baby names: ROBERT, JANE, MARIA, PETER, and ETERNIA (ignoring their gender suitability), we give some examples of the modified queries (this time the query only has one parameter) below:

- `Query(‘‘ER’’)` = 3 as we have ROBERT, PETER, and ETERNIA.
- `Query(‘‘E’’)` = 4 as we have ROBERT, JANE, PETER – there are two character 'E' in 'PETER', but we only count him once and ETERNIA – same here.
- `Query(‘‘IA’’)` = 2 as we have MARIA and ETERNIA.
- `Query(‘‘ROBERT’’)` = 1 as we only have one ROBERT.
- `Query(‘‘ETER’’)` = 2 as we have PETER and ETERNIA.

Note that your solution for the original problem will be wrong for this R-option requirement. Therefore, students who want to attempt this R-option has to submit `BabyNamesR.java` instead of `BabyNames.java`. As the time needed to complete the R-option can be significant, non CS2010R students can choose to attempt the R-option only (and get 110 points if it is correct) or choose to go safe with the normal PS1 (and get at most 100 points).

Another note: Partial marks will be given for incomplete solution of this R-option.

[3]A substring of a string $T = T_0T_1...T_{n-1}$ is string $P = T_iT_{i+1}...T_j$ where $0 \leq i \leq j < n$.