Problem report metrics calculator (updated on 2019/6/11)

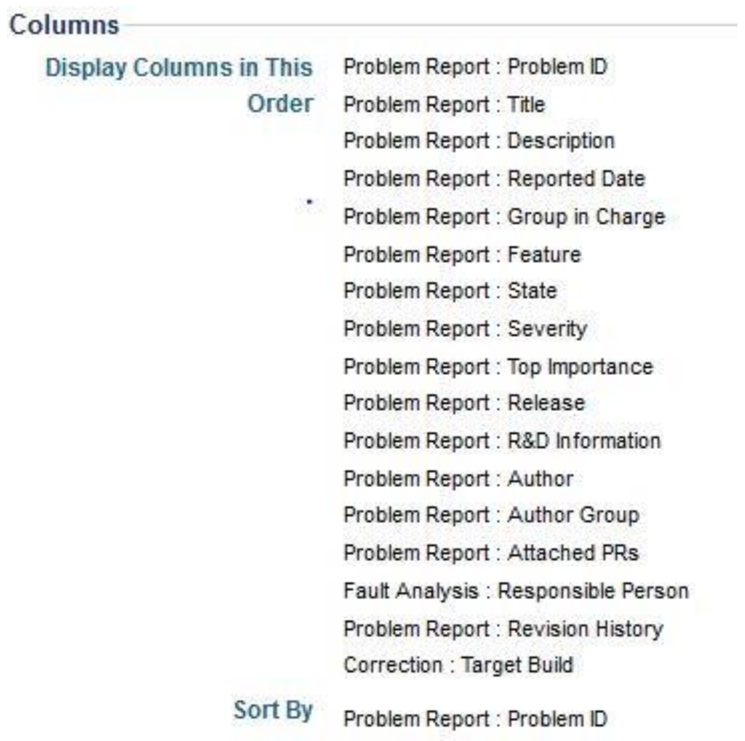
1. Introduction

Nokia uses pronto system to track problems identified in various testing activities that are created by various R&D groups, which is called as "Group In Charge", a.k.a GIC. Each GIC belongs to one unique R&D tribe, while one tribe may own more than one GICs for various reporting and management reasons. Different problems that are reported by various testing teams might share the same root cause, such that those problems might be attached together to reduce tracking and management efforts.

The challenge here is to analyze the pronto activity data and calculate desired information for further processing.

1.1. Input

Pronto system provides export functionality that can provide records as csv format (comma separated text file), so users can open them as Excel sheet. The CSV file is exported and downloaded from pronto statistic page and contains below fields (in the order as specified below):



Columns

Display Columns in This Order	Problem Report : Problem ID
	Problem Report : Title
	Problem Report : Description
	Problem Report : Reported Date
	Problem Report : Group in Charge
	Problem Report : Feature
	Problem Report : State
	Problem Report : Severity
	Problem Report : Top Importance
	Problem Report : Release
	Problem Report : R&D Information
	Problem Report : Author
	Problem Report : Author Group
	Problem Report : Attached PRs
	Fault Analysis : Responsible Person
	Problem Report : Revision History
	Correction : Target Build
Sort By	Problem Report : Problem ID

Sample file is manually truncated to verify a single criteria of this challenge and doesn't contain "malformed free text" before the normal CSV header. While actual pronto statistic reports may produce a few lines of extra free texts before the formal csv table is found.

To obtain perfect score, there is a final challenging yet real report **which contains ~50MB data as input.** Performance (execution time) would be measured based on those real data set that contains **all possible combinations of below mentioned sub-challenges** - you have to produce correct results first to ensure your performance data is considered. Also make sure your csv parser is able to take below cases into account:

- Extra few lines of free texts before the normal CSV header begins
- Header fields might be wrapped in double quotas, like "Problem ID"
- One single field might span more than one line so new line ("\n" or "\r\n") has to be processed properly
- Commas might be part of wrapped field content, like "some thing, another, etc"
- A certain line might be extremely long (>1024 characters for example)

There will be multiple inputs provided from easier to harder ones, so your score would be calculated based on how many sub-challenges are conquered.

1.2. Output

For each sub-challenge (including the final big one), an unique JSON output should be generated, which contains calculated information that would facilitate further processing.

The structure of output json files should be organized like below

```
{
  "records" : [
    {
      "PR ID": "PR349542",
      "Group In Charge": "NIESSCBTSMZOAM",
      "Reported Date": "06.06.2018",
      "Author": "Chen, Jackie 2. (NSB - CN/Hangzhou)",
      "Author Group": "NIHZSIV2",
      "State": "Closed",
      "Transfer Path": [
        "NIESSCBTSMZOAM"
      ],
      "Transfer times": 1,
      "Solving Path": [
        "NIESSCBTSMZOAM"
      ],
      "Pingpong times": 0,
      "Attached PRs": "PR352628",
      "Attached": "no",
      "Attached To": ""
        },
    {
      "PR ID": "PR349552",
      "Group In Charge": "NIESSCBTSMZOAM",
      "Reported Date": "06.06.2018",
      "Author": "Shentu, Junda (NSB - CN/Hangzhou)",
      "Author Group": "SRANIVHZ_NEVE",
      "State": "Correction Not Needed",
      "Transfer Path": [
        "NIESSCBTSMZOAM"
      ],
      "Transfer times": 1,
      "Solving Path": [
        "NIESSCBTSMZOAM"
      ],
      "Pingpong times": 0,
      "Attached PRs": "< empty >",
      "Attached": "no",
      "Attached To": ""
    },
  ],
  "total" : "2"
}
```

Some of those output fields of a given record is a direct copy of input records, this is typically needed for next step analysis or human inspection. Other fields should be calculated by more calculation as specified in following sections.

1. Number of output records should be exactly the same as input records.
2. Different records should be organized by ascending order with regard to "PR ID" field.

2. Challenges Requirements

2.1. Transfer Analysis

A problem might be transferred among different groups of teams for resolution, we need to

- Exclude transfers inside same organization, like RD1_TeamA → RD1_TeamB should be stripped from results (they are meaningless transfers), **this filtering should be performed firstly** before all other calculations.
- Duplicate transfers should be merged if there ever exist, which means that 2 **consecutive** same transfers would be considered as one.
- Calculate the transfer path of remaining meaningful transfers.

Transfer relations shall be parsed and extracted from the "Revision History" field - note there might be various patterns due to complexity of actual R&D activities

- A PR can be transferred back and forth many times (as ping-pong)

- A PR can never be transferred before closed - always touched by same group, in this case there might be no transfer records at all
- A PR can be transferred **by organization internal arrangements** - in this case it is actually a transfer noise, so we want to **exclude them**
- If there's a transfer path, the groups that touched this PR **should be in strict order as was processed from earliest to latest**
- The field "transfer times" counts how many times a given PR was ever transferred plus one, so **"1" means no transfer** at all.

Below groups relation are considered in this challenge (can be extended per further requirements)

- Parent group: ECE_DEV_FOU
- Child group_1: ECE_DEV_FOU_OAM_MZ
- Child group_2: ECE_DEV_FOU_OAM_MZ_EXT
- Child group_3: MANO_HZH_MZOM
- Child group_4: MANO_HZH_MZOM_EXT
- Child group_5: ECE_DEV_FOU_UP_L2_HI
- Child group_6: ECE_DEV_FOU_UP_L2_LO

2.1.1. Sub-challenge on calculating ping-pong

Detect ping pong and calculate number of ping-pongs. Actual ping-pong rules can be hard to reason especially when we consider also indirect ping-pongs. Our challenge here is to try to identify below scenarios

- Problem is transferred back to previous touched group, for example

```
A -> B #okay
B -> A #ping pong
A -> B #ping pong again to predecessor?
```

- Problem is transferred to a group that has ever (indirectly) touched this problem before

```
A -> B #okay, path = [A, B]
B -> C #okay, path = [A, B, C]
C -> A #ping pong back as A has processed!, path = [A]
A -> D #okay, path = [A, D]
D -> C #ping pong back as C has processed!, path = [A, C]
C -> F #okay, path = [A, C, F]

Soving path: A->C->F
```

The field "Solving path" shows the actual problem solving path excluding ping-pong, while file "pingpong times" gives the number of ping-pong calculated.

- The path shown in solving path **may not be a real path** in original transfer history, since we revert back all those processed groups and consider them as a ping-pong back.
- To simplify the calculation, when checking ping pong back, we **do NOT consider** transfer back to another group in previously touched organization.

Example transfer path:

```
"Transfer Path": [
        "NIWRSRF",
        "5GRAN_US_SYSTEM_INTRODUCTION",
        "NIWRSRF",
        "MANO_GENERAL",
        "MANO_MNL_RADIOCTRL",
        "NIWRSRF",
        "MANO_GENERAL",
        "MANO_MNL_RADIOCTRL",
        "NIWRSRF",
        "MANO_GENERAL",
        "NIWRSRF",
        "5G_L1",
        "5G_L1_SW",
        "BB_PSW_LFS",
        "BB_PSW1_PS_LFS",
        "TRS_5G_ASIK_GENERAL",
        "BB_PSW1_PS_LFS",
        "BB_PSW_FDSWSWIM",
        "BB_PSW1_PS_LFS"
],
```

Would yield below outputs:

```
      "Solving Path": [
        "NIWRSRF",
        "5G_L1",
        "5G_L1_SW",
        "BB_PSW_LFS",
        "BB_PSW1_PS_LFS"
      ],
      "Pingpong times": 9,
```

See also original information from https://pronto.inside.nsn.com/pronto/problemReportSearch.html?freeTextdropDownID=prId&searchTopText=CAS-193006-F8T7

2.2. Attach and Uniqueness

Multiple problems reported might be caused by one same root cause such that they might be attached, we need to

- Consider all attached items as one problem
- Select **one item by giving smallest ID** to denote the whole group of problems, we call it a **representative record**
- Attached items may **not be consistent in the raw report**, like a → b, b → c, c → b would yield one single item so they are treated as one record.

There are 3 fields in output JSON object structure regarding the attach checking

- "Attached PRs" should copy out the original record's attach information
- Another 2 fields "Attached" and "Attached To" are set per attach calculation result
    - If given PR is calculated as unique **representative** of the attach groups of records, "Attached" should be set as "no", and "Attached To" should be set as "".
    - if given PR is calculated as inside a group of attach relations but not the unique **representative** record, "Attached" should be set as "yes", and "Attached To" should contain the **representative** record "PR ID".
    - If given PR has never be attached with any other PRs, "Attached" should be "no" and "Attached To" should be set as ""

2.2.1. Sub-challenge on dangling attaches

The input records are exported from the system without prior knowledge of whether an earlier attached record is captured by provided criteria, it is also possible that a certain attaching source problem can't be found in input records, we can it a "dangling attach". We don't want to designate a dangling record as the representative record, so it should be excluded from representative record selection.

2.3. Combination of sub-challenges

Above sub-challenges can be combined in both small inputs and final inputs. Besides the final big challenge, below sub-challenges are checked to calculate your score

1. Simple parsing check - no transfer, no attach ever exists in inputs
2. Simple transfer without ping-pong relation, no attach relation exists in inputs
3. Complex transfer with ping-pong, but no attach relation exists in inputs
4. Combined both ping pong, no ping-pong and even no transfer, but no attach relation inputs
5. Simple attach but no transfer exists, no dangling attaches inputs
6. Attach with dangling attach, but no transfer exists in inputs
7. Attach with transfer but no ping-pong exists, no dangling attaches exists in inputs
8. Attach with transfer and ping-pong exists, no dangling attaches exists in inputs
9. Combined attach and ping pong co-exists, no dangling attaches exists in inputs

Final big challenge requires

1. Full csv exports with malformed CSV free text lines
2. Contains all possible combinations of above sub-challenges

3. Performance would be measured when correctness is guaranteed

3. Additional Notes

1. Pretty json output is preferred but not required, such that differences in tabs, spaces are considered as irrelevant. Of course if those special white space characters are part of record name or object values, you should never omit them.
2. Output json should keep the records as in sorted manner, otherwise you'll get a penalty of 20% deduction.
3. Non-important fields missing wouldn't be considered as fatal-error, so you'll still get some score if you forget to copy out those non-calculated fields. For detailed list of calculated fields, please refer to above specification.
4. All calculated fields should match exactly with desired outputs, otherwise you'll get 0 score for a given specific challenge.
5. Performance measurement would only be scored by the final complete input - be care for that input file is quite big with thousands of input records.

4. Examples (updated on 2019/6/4)

http://10.182.198.199/Problem1/sample.zip

NOTE

The input of your program is 1 argument, the path of a csv file exported from PRONTO system.

The output is standard output print, the format is JSON as the examples.