Test automation results categorization aggregation

In our test automation, a large number of execution records are generated. We store the execution records in row one by one.

the information of storage are typically contain following field:
id: the only index number which means execution record.
testCase_id: the case id number,
team_id: the case belonging to which team.
build_id: the tested software version number.
execution_time: test case execution time.
result: the result of the execution test case (1 means case pass, 2 means case fail).
phase_id: the test phase id.

We store these bulk execution results in CSV files. sample as the following table:

id	testCase_id	build_id	team_id	execution_time	result	phase_id
2446322	18000	1140	1	2019/4/1 0:04	1	1
2446324	18001	1140	1	2019/4/1 0:11	2	1
2446325	18002	1140	2	2019/4/1 0:21	2	1
2446326	18002	1140	2	2019/4/1 0:32	1	1
2446327	18002	1140	2	2019/4/1 1:02	2	1
2446328	18003	1140	2	2019/4/1 1:11	1	1
2446333	18003	1140	2	2019/4/1 1:23	2	1
2446334	18001	1141	1	2019/4/2 3:12	1	1
2446335	18004	1141	2	2019/4/3 3:02	1	2
2446336	18005	1141	1	2019/4/3 5:25	2	2
2446343	18006	1141	2	2019/4/3 5:36	1	2
2446344	18007	1141	2	2019/4/3 7:12	2	2
2446345	18001	1141	1	2019/4/3 6:39	1	2
2446346	18000	1141	3	2019/4/3 6:45	2	2

2446347	18000	1141	3	2019/4/3 6:45	2	2

The above is the original execution record data sample, but most of us need aggregation statistics, and the statistical display style needs to be as follows:

build	phase	team_num	case_num	team_id	pass_num	fail_num	case_num	pass_rate
1140	1	2	4	1	1	1	2	50%
				2	2	0	2	100%
1141	1	1	1	1	1	0	1	100%
	2	3	6	2	2	1	3	67%
				1	1	1	2	50%
				3	0	1	1	0%

The json file corresponding to this statistical style is as below, which is the final answer we need for the original csv file


```
[
  {
    "phase": [
      {
        "phase_id": "1",
        "team_num": 2,
        "case_num": 4,
        "test_info": [
          {
            "case_num": 2,
```

```
          "team_id": "1",
          "execution_passed": 1,
          "execution_failed": 1,
          "pass_rate": "50%"
        },
        {
          "team_id": "2",
          "case_num": 2,
          "execution_passed": 2,
          "execution_failed": 0,
          "pass_rate": "100%"
        }
      ]
    }
  ],
  "build_id": "1140"
},
{
  "phase": [
    {
      "phase_id": "1",
```

```
  "team_num": 1,
  "case_num": 1,
  "test_info": [
    {
      "case_num": 1,
      "team_id": "1",
      "execution_passed": 1,
      "execution_failed": 0,
      "pass_rate": "100%"
    }
  ]
},
{
  "phase_id": "2",
  "team_num": 3,
  "case_num": 6,
  "test_info": [
    {
      "case_num": 3,
      "team_id": "2",
      "execution_passed": 2,
```

```
        "execution_failed": 1,
        "pass_rate": "67%"
      },
      {
        "team_id": "1",
        "case_num": 2,
        "execution_passed": 1,
        "execution_failed": 1,
        "pass_rate": "50%"
      },
      {
        "team_id": "3",
        "case_num": 1,
        "execution_passed": 0,
        "execution_failed": 1,
        "pass_rate": "0%"
      }
    ]
  }
],
"build_id": "1141"
```

```
  }
]
```

Please observe the above sample and summarize the rules for converting statistics, and then programmatically to generate the desired json result from the original CSV file.

Notes: we have some simple test cases, these simple test cases already include all the aggregation rule, you can use these cases to verify the correctness of your program. You need to print out the result in json format in standard output. You can down load these cases from the link below.

http://10.182.198.199/Problem2/sample.zip

In the formal verification, beside the simple cases, we also large CSV files for performance test, the large file contains more than 300,000 lines of data.

Input (as the only argument of your program): the path of a CSV file

Output: standard output print in json format