相似BUG报告计算说明

作者：韩禧

结构分三层：

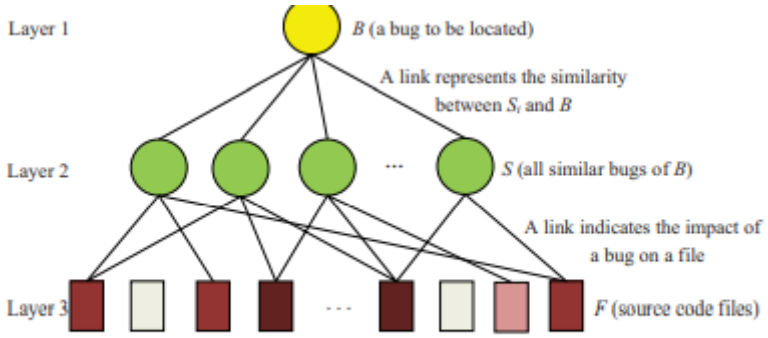

Figure 3. Heterogeneous Bug-File graph

Layer1代表现有的需要定位的bug

Layer2代表已经被解决（fixed），且和现有bug报告相似的历史bug

Layer3代表与Layer2中bug相关联的代码文件

第一步

- 参数1：当前bug的bug报告的**向量**bug_report
- 参数2：之前已经fix的bug报告的**向量组**fixed_bug_report_list
- 结果：weight_list，重构后为一维字典，以reportId为key，以bug报告的权重值（float）为value

计算bug报告的相似度，使用cos(q,d)进行计算：

$$Similarity(q,d) = \cos(q,d) = \frac{\overrightarrow{V_q} \bullet \overrightarrow{V_d}}{\left|\overrightarrow{V_q}\right|\left|\overrightarrow{V_d}\right|}$$

并保留与当前bug相似度为正的已修复的bug报告，将计算得到的cos(q,d)作为权重weight，构成weight_list（列表或字典，如果是字典的话以reportId为key，以权重值（float）为value）

在原代码中为了构成和bug报告顺序对应的list，将非正的权重值记为了0

```
for report in bug_report_list:
    simi_count = get_cos_sim(bug_report, report)
    if simi_count > 0:
        weight_list.append(simi_count)
    else:
        weight_list.append(0)
```

第二步

- 参数1：weight_list，记录了bug报告的权重，结构为字典，以reportId为key，以权重值（float）为value
- 参数2（重构后与原数据结构不同）：bug_report_list，记录了bug报告对应的codefile，数据结构可以使用一维字典，key为reportId，value为对应的codefile的list
- 结果：simi_scores，二维字典，simi_scores的key为reportId，value为对应的源码文件得分的字典simi_scores[reportId]，而字典simi_scores[reportId]的key为codefile（str），value为源码文件的得分（float）

计算源代码文件的相关度（simi_scores）

对于每个已经修复的bug报告，与它相关联的源代码文件平分它的权重。

而每个源代码文件的得分就会与所有关联到它的bug报告有关：

$$SimiScore = \sum_{\substack{\text{All } S_i \text{ that} \\ \text{connect to } F_j}} (Similarity(B, S_i) / n_i)$$

这个公式是对源码文件相似得分的计算，Similarity(B,Si)代表了B和Si的相似度，B代表现在的bug，Si代表已经修复的历史bug，ni代表与Si相关的源代码文件数量。将所有的平均值加和就是每个源码文件的最终得分，Fj代表当前的源代码文件，SimiScore就是Fj的相关度得分

原代码中采用字典记录源代码文件，按照bug报告为序进行遍历，最后可以得到每个相关联的源码文件的得分，以下为demo代码，并非最终实现：

```
# 计算关联的权值
simi_scores = {}
ordinal = 0
for code_list in source_code:
    if weight_list[ordinal] == 0:
        continue
    n = len(code_list)
    for file in code_list:
        simi_scores[file] += weight_list[ordinal] / n
    ordinal += 1
JSONWriter(simi_score_path, simi_scores).writeFile()
```

其中，

- simi_scores是用于记录源代码文件的得分的字典结构
- 如果bug报告的权重值为0就跳过
- source_code是存放每个bug报告的关联代码文件的二维list， code_list是当前bug报告对应的源代码文件的list
- 遍历加和后得到每个**源码文件**的得分（simi_scores)

最后对结果使用JSON存储，simi_score_path，并在result_printer.py中进行了调用，用于计算final_score