DSA Final Project

Team No.65
B08501098 柯晨緯 B08703118 黃紹宇 B06502006 錢逸魁

1 Data Structures & Algorithms

- Expression Match

  We used Stacks and Bitset in this query. The reason we use bitset is that we can record which mail used a specific word by storing mail Ids in the corresponding bit, and we can match which mail satisfies the expression or not in $O(1)$.

  We've tried two implementations.

  1. Method 1: We use two stacks to store data, one for operators, the other for operands. This method only needs to scan the expression from head to tail once. If we encounter an operand, we get it's hash value via Trie-Search, and push the corresponding bitset of mails into the operand stack; if we encounter an operator, we need to consider the precedence of this operator to determine how far we can calculate in the stacks. After scanning the expression, we calculate the leftovers in the two stacks until the operand stack have only one bitset left. This method is O(length of expression). (Scanning the expression from head to tail is O(length of expression), and calculating in stacks is O(length of expression), too.)
  2. Method 2: We first convert the infix expression to postfix, by using a stack and an array. After we convert the expression into postfix, we just need to scan the operands and the operators in the array, calculate them or push them into a stack of bitsets until the end of the array. This method is also O(length of expression), too. (Translating into postfix needs O(length of expression), and calculating is O(length of expression), too.

  After testing the two implementation, we found that the second method is about 120% faster than the first one. The reason of being faster is maybe the if-else condition in the second method is lesser than the first one, so every time we do this query, we spend less time on the if-else statement, thus increases the reward.

- Find Similarity

  In this task, we've tried two implementations.

  1. Method 1: We use bitset to record the content of each mail. When calculating the similarity, we directly do And-Or operations on every pair of mails. The time complexity of each pair comparison is O(number of tokens).
  2. Method 2: Because the total number of token is far from affordable (about 140000), we attempt to use list to record the content of each mail. By evaluating the list length of each mail, we know that the maximum is about 4000, which is way below

than the size of bitset. We do construct bitset same as Method 1, but now we compare the bitset with element in each list. By this technique, the time complexity of this method becomes O(length of list), which faster than Method 1.

- Group Analyses

  We've tried three implementations.

  1. Method 1: Using DFS with adjacency list to visit every node in the graph. By keeping track of how many time we call DFS() on a parent node, we can know how many components are in the graph. This operation can be done in O(V + E). V is the number of vertices and E is the number of edges.
  2. Method 2: Array based disjoint-set union by rank with path compression. We do make set on every new node and keep track of it. Then union every edge and keep track of every success union operation and the size of the largest component. This operation can be done in roughly O(M). M is the number of make set operation.
  3. Method 3: Array based disjoint-set. This method is similar to method 2, but slower when the size of component is big. This operation can be done in roughly O(M + N*N). M is the number of make set operation and N is the size of component.

  We choose to use Method 3 because Method 1 is recursive therefore time consuming and the size of components aren't that big so doing rank comparison and path comparison do more harm than good.

2 Cost Estimations of Queries

Query Types	Number of answering times (in given interval)	Expected Points Gained
Expression Match	317741	317741
Find Similar	2985	238800
Group Analyse	1988607	397721.4

Figure 1: Run Time & Reward Table

- The numbers we provide in the table is just for reference, and not absolute accurate. The error might come from different factors, such as the test data that we got may be smaller or weaker, or our computers and the judge have different environment and different computing power. We just use this table to see which query might have a better profit.

3 Scheduling Strategy

- We decide to prioritize on the reward of each query because we optimize our code to perform better when the quantity of data to dealt with is large, which usually comes with higher reward, so it's more efficient for us to answer those with higher reward only.

  Our strategy is to find the optimal thresholds of reward for all three kinds of query. In our tests, group analyses can get the most point within given time. So we didn't set a

threshold for group analyses as for the other two query, we find the threshold through trial and error. In our finding, the threshold for expression match is >1.28 and for data find similar is >104.2.

4 Additional Notes

- In Find Similarity, we found that a mail's similarity will have an upper and lower bounds. Knowing this property, we can calculate the upper and lower bounds offline, and when doing Find Similarity, we can first check if the required similarity is in the range. If it is too high, then it will never satisfy the required similarity; if it is too low, we can just put every mail except itself into the answer array. With the help of the precalculated bounds, we just need to calculate those required similarity is in the interval.