**Code Jam 2008 - Qualification Round**

# Analysis: Saving the Universe

The first task in the new Google Code Jam is about, no surprise, search engines, and also, the grandiose feat of saving the universe in a parsimonious way. However, putting all the fancies aside, the problem itself is easy.

List all queries one by one, and break them up into segments. Each segment will be an interval where we use one search engine, and when we cross from one segment to another we will switch the search engine we use. What can you say about each segment? Well, one thing for sure is:

> Never ever ever have all **S** different search engines appear as queries in one segment. (*)

Why is this? Because if all **S** names appeared in one segment, then any search engine used for that segment will encounter at least one query that is the same as its name, thus exploding the universe!

Working in the opposite direction, (*) is all we need to achieve; as long as you can partition the list of queries into such segments, it corresponds to a plan of saving the universe. You don't even care about which engine is used for one segment; any engine not appearing as a query on that segment will do. However, you might sometimes pick the same engine for two consecutive segments, laughing at yourself when you realize it; why don't I just join the two segments into one? Because your task is to use as few segments as possible, it is obvious that you want to make each segment as long as possible.

This leads to the greedy solution: Starting from the first query, add one query at a time to the current segment until the names of all **S** search engines have been encountered. Then we continue this process in a new segment until all queries are processed.

Sample code in C++, where **st** is the set of queries in the current segment, **q** is the next query, and **count** is the number of switches.

```
st.clear();
count = 0;
for (int i=0; i<Q; i++) {
  getline(cin, q);
  if (st.find(q) == st.end()) {
    if (st.size() == S-1) {
      st.clear();
      count++;
    }
    st.insert(q);
  }
}
```

If **st** is a hashset, you expect the solution run in O(n) time. Note that this solution uses the fact that each query will be a search engine name and so we can ignore the list of names provided in the input.

Let us justify that the greedy approach always gives the optimal answer. Think of the procedure as Q steps, and we want to show that, for each i, there is (at least) one optimal choice which

agrees with us on the first i steps. We do this inductively for i = 0, then i = 1, and so on. The proposition for i = Q when proven true will imply that our algorithm is correct.

So, the key points in the inductive step i:

1. If adding the next query will explode the universe, we must start a new segment. Any optimal choice agrees with us for the first (i-1) steps must also do that.
2. If adding the next query will not explode the universe, we do not start a new segment. We know there is an optimal solution R agreed with us for (i-1) steps. Even if in R a new segment is started at step i, we can modify it a little bit. Let R' be the plan that agrees with R, but instead of starting a new segment on the i-th step, we delay this to the (i+1)st. It is clear that R' will also make the universe safe, and has no more switches than R does. So, R' is also an optimal solution, and agrees with our choice for the first i steps.

The similar lines of justifications work for many greedy algorithms, including the well beloved minimum spanning trees.