USACO Traingate 'hidden' Analysis

by Alex Schwendner

First, we concatenate two copies of our string S, yielding SS. This way, instead of dealing with cyclic shifts, we simply want the lexicographically least L-letter substring of SS. But as the remaining letters of SS after an L-letter substring are just the same as those at the start of the substring, we might as well search for the lexicographically least suffix of SS. If we append a terminating character which is lexicographically after z, we will always choose the first appropriate substring in SS.

How do we efficiently find the lexicographically least suffix of SS?

For each i, let V[i] be the length of the longest substring starting at i which is the lexicographically least substring in SS of that length. For example, in "abcabab", the lexicographically least 2-letter substring is "ab", the least 3-letter substring is "aba", so V[0]=2, as the suffix starting at 0 goes "abc...".

We initialize V to 0, and make a list of all possible starting points. We go through the list, and for each starting point i in our list, we consider V[i+V[i]]. If V[i+V[i]] > 0, then as the V[i+V[i]]-letter substring starting as position i+V[i] is minimal, and as the V[i]-letter substring starting at i is minimal, the (V[i]+V[i+V[i]])-letter substring starting at i is minimal. (We're joining our current substring to a previously computed substring.)

If V[j+V[j]] > V[i+V[i]] for some j in our list, then the suffix starting at j is less than the suffix starting at i, as they agree on the first V[i]=V[j] letters (V[x] is the same for all x in our list), and j has more minimal letters afterwards then i does (V[j+V[j]] > V[i+V[i]]). Thus, for all i in our list such that V[i+V[i]] is maximal, we set V[i]=V[i]+V[i+V[i]].

We also set a flag for i+V[i] so that if it's also in our list, we don't process it ever again, as the prefix starting at i is better. This prevents the strings in our list from overlapping. (As we're not processing i+V[i] anymore, we may later have a wrong value for V[i+V[i]], but this doesn't matter as we'll never be using it again, as the string at i+V[i] is inside of the new one at i.) We remove all i for which V[i+V[i]] is not maximal from our list. If, on the other hand, V[i+V[i]] = 0 for all i in our list, then we simply add 1 to V[i] for all i for which S[i+V[i]] is minimal over all i in our list, and we remove other i from our list.

We repeat this process until our list has only 1 position in it, which will be our answer.

What is the runtime of our algorithm? Well, at each iteration, for each i in our list, we either merge two substrings, removing an i from our list, or we extend a substring by 1 or more letters. Both of these are constant time operations. Because we use previous V values when extending substrings, we can walk over a given letter at most once. We can also remove an i from our list only once. As we have 2L items in our list, and 2L letters, our algorithm is O(L).

```
#include <string.h>
#include <stdio.h>

const int MAXN = 2*100000+1;

int n;
char s[MAXN];
int v[MAXN];

int k1,k2;
int L1[MAXN],L2[MAXN];
int *l1=&L1[0], *l2=&L2[0];

int main(){

  FILE *fin = fopen("hidden.in", "r");
  fscanf(fin, "%d", &n);
  for(int i = 0; i < n; i+=72){
    fscanf(fin, "%s", &s[i]);
  }
  fclose(fin);

  memcpy(&s[n], &s[0], n);
  s[2*n] = 'z'+1;
  n = 2*n+1;

  for(int i = 0; i < n; ++i) v[i] = 0;
  for(int i = 0; i < n; ++i) l1[i] = i;
  k1 = n;

  while(k1 > 1){
    char least = 'z'+1;
    int most = 0;
    for(int i = 0; i < k1; ++i){
      least <?= s[l1[i]+v[l1[i]]];
      most  >?= v[l1[i]+v[l1[i]]];
    }

    k2 = 0;
    if(most > 0){
      for(int i = 0; i < k1; ++i){
        if(v[l1[i]] != -1 && v[l1[i]+v[l1[i]]] == most){
          l2[k2++] = l1[i];
          v[l1[i]+v[l1[i]]] = -1;
          v[l1[i]] += most;
        }
      }
    } else {
      for(int i = 0; i < k1; ++i){
        if(v[l1[i]] != -1 && s[l1[i]+v[l1[i]]] == least){
          l2[k2++] = l1[i];
          v[l1[i]] += 1;
        }
      }
    }

    int *ls = l1;
    l1 = l2;
    l2 = ls;
    k1 = k2;
  }

  FILE *fout = fopen("hidden.out", "w");
  fprintf(fout, "%d\n", l1[0]);
  fclose(fout);

  return(0);
}
```