String Algorithms

The Longest Duplicated String
Markov Text

Longest Duplicated String

The Problem

Input: ‘‘Ask not what your country can do for you, but what you can do for your country’’.

Output: ‘‘ can do for you’’ (15 chars)

A Simple Algorithm (at least quadratic)

```
maxlen = -1
for i = [0, n)
    for j = (i, n)
        if (l=comlen(&c[i], &c[j]))
                > maxlen
            maxlen = l
            maxi = i
            maxj = j
```

An Important Function

```
int comlen(char *p, char *q)
    i = 0
    while *p && (*p++ == *q++)
        i++
    return i
```

A Fast Algorithm

Key Data Structures

```
#define MAXN 5000000
char c[MAXN], *a[MAXN];
```

Input ‘‘Suffix Array’’ for ‘‘banana’’:

```
a[0]: banana
a[1]: anana
a[2]: nana
a[3]: ana
a[4]: na
a[5]: a
```

The Sorted Suffix Array

```
a[0]: a
a[1]: ana
a[2]: anana
a[3]: banana
a[4]: na
a[5]: nana
```

Scan through to find longest duplicated string (‘‘ana’’).

The Suffix Array Code

The Code (usually about $O(n \log n)$)

```
while (ch = getchar()) != EOF
    a[n] = &c[n]
    c[n++] = ch
c[n] = 0
qsort(a, n, sizeof(char *), pstrcmp)
for i = [0, n)
    if comlen(a[i], a[i+1]) > maxlen
        maxlen = comlen(a[i], a[i+1])
        maxi = i
printf("%.*s\n", maxlen, a[maxi])
```

4.8 secs on 807,503 characters of Homer's *Iliad*:

> whose sake so many of the Achaeans have died at Troy, far from their homes? Go about at once among the host, and speak fairly to them, man by man, that they draw not their ships into the sea.

Markov English Letters

Monkey Typing: uzlpcbizdmddk njsdzyyvfgxbgjjgbt-sak rqvpgnsbyputvqqdtmgltz ynqotqigexjumqphu-jcfwn ll jiexpyqzgsdllgcoluphl sefsrvqqytjakmav

Order-0: saade ve mw hc n entt da k eethetocusos-selalwo gx fgrsnoh,tvettaf aetnlbilo fc lhd okleut-sndyeoshtbogo eet ib nheaoopefni ngent

Order-1: t I amy, vin. id wht omanly heay atuss n macon aresethe hired boutwhe t, tl, ad torurest t plur I wit hengamind tarer-plarody thishand.

Order-2: Ther I the heingoind of-pleat, blur it dwere wing waske hat trooss. Yout lar on wassing, an sit." "Yould," "I that vide was nots ther.

Order-3: I has them the saw the secorrow. And win-tails on my my ent, thinks, fore voyager lanated the been elsed helder was of him a very free

Order-4: His heard." "Exactly he very glad trouble, and by Hopkins! That it on of the who difficentralia. He rushed likely?" "Blood night that.

Markov English Words

A finite-state Markov chain with stationary transition probabilities.

Order-1: The table shows how many contexts; it uses two or equal to the sparse matrices were not chosen. In Section 13.1, for a more efficient that ‘‘the more time was published by calling recursive structure translates to build scaffolding to try to know of selected and testing and more robust

Order-2: The program is guided by verification ideas, and the second errs in the STL implementation (which guarantees good worst-case performance), and is especially rich in speedups due to Gordon Bell. Everything should be to use a macro: for $n = 10,000$, its run time;

Order-3: A Quicksort would be quite efficient for the main-memory sorts, and it requires only a few distinct values in this particular problem, we can write them all down in the program, and they were making progress towards a solution at a snail’s pace.

Algorithms and Programs

Shannon, 1948: ‘‘To construct [order-1 letter-level text] for example, one opens a book at random and selects a letter at random on the page. This letter is recorded. The book is then opened to another page and one reads until this letter is encountered. The succeeding letter is then recorded. Turning to another page this second letter is searched for and the succeeding letter recorded, etc. A similar process was used for [order-1 and order-2 letter-level text, and order-0 and order-1 word-level text]. It would be interesting if further approximations could be constructed, but the labor involved becomes enormous at the next stage.’’

Kernighan and Pike’s Exposition, 1999

Build a data structure while training

Randomly traverse the structure to generate

Implemented in C, C++, Java, Awk, Perl

Suffix Arrays to the Rescue

Word array for $k = 1$ "of the people, by the people, for the people":

```
word[0]: by the
word[1]: for the
word[2]: of the
word[3]: people
word[4]: people, for
word[5]: people, by
word[6]: the people,
word[7]: the people
word[8]: the people,
```

The Critical Function

```
int wordncmp(char *p, char* q)
    n = k
    for ( ; *p == *q; p++, q++)
        if (*p == 0 && --n == 0)
            return 0
    return *p - *q
```

Code Sketch, Part 1

Read and Store Training Sample

```
word[0] = inputchars
while scanf("%s", word[nword]) != EOF
    word[nword+1] = word[nword] +
                    strlen(word[nword]) + 1
    nword++
/* put k null characters at end */
for i = [0, k)
    word[nword][i] = 0
```

Print First *k* Words

```
for i = [0, k)
    print word[i]
```

Sort The Array

```
qsort(word, nword, sizeof(word[0]), sortcmp)
```

Code Sketch, Part 2

Generate Text

```
phrase = first phrase in input array
loop
    perform a binary search for phrase
        in word[0..nword-1]
    for all phrases equal in k words
        select one at random,
            pointed to by p
    phrase = word following p
    if k-th word of phrase is length 0
        break
    print k-th word of phrase
```

Pseudocode for Text Generation

```
phrase = inputchars
for (left = 10000; left > 0; left--)
    l = -1
    u = nword
    while l+1 != u
        m = (l + u) / 2
        if wordncmp(word[m], phrase) < 0
            l = m
        else
            u = m
    for (i = 0; wordncmp(phrase, word[u+i])
                == 0; i++)
        if rand() % (i+1) == 0
            p = word[u+i]
    phrase = skip(p, 1)
    if strlen(skip(phrase, k-1)) == 0
        break
    print skip(phrase, k-1)
```

Comparison to Typical Approaches

Similar speed, less space, less code

The Complete Code

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

char inputchars[4300000];
char *word[800000];
int nword = 0;
int k = 2;

int wordncmp(char *p, char* q)
{       int n = k;
        for ( ; *p == *q; p++, q++)
                if (*p == 0 && --n == 0)
                        return 0;
        return *p - *q;
}

int sortcmp(char **p, char **q)
{       return wordncmp(*p, *q);
}

char *skip(char *p, int n)
{       for ( ; n > 0; p++)
                if (*p == 0)
                        n--;
        return p;
}

int main()
{       int i, wordsleft = 10000, l, m, u;
        char *phrase, *p;
        word[0] = inputchars;
        while (scanf("%s", word[nword]) != EOF) {
                word[nword+1] = word[nword] + strlen(word[nword]) + 1;
                nword++;
        }
        for (i = 0; i < k; i++)
                word[nword][i] = 0;
        for (i = 0; i < k; i++)
                printf("%s0, word[i]);
        qsort(word, nword, sizeof(word[0]), sortcmp);
        phrase = inputchars;
        for ( ; wordsleft > 0; wordsleft--) {
                l = -1;
                u = nword;
                while (l+1 != u) {
                        m = (l + u) / 2;
                        if (wordncmp(word[m], phrase) < 0)
                                l = m;
                        else
                                u = m;
                }
                for (i = 0; wordncmp(phrase, word[u+i]) == 0; i++)
                        if (rand() % (i+1) == 0)
                                p = word[u+i];
                phrase = skip(p, 1);
                if (strlen(skip(phrase, k-1)) == 0)
                        break;
                printf("%s0, skip(phrase, k-1));
        }
        return 0;
}
```