Basic Mathematical Techniques for Computer Scientists – a few words on ~~words~~ strings

Paweł Gawrychowski

December 17, 2012

A word is simply a sequence of characters over some fixed alphabet. We denote that by writing $w \in \Sigma^*$, which read "w is a word consisting of a number of characters from alphabet Σ".

$$\begin{aligned} aababaaab &\in \{a,b\}^* \\ 01101110 &\in \{0,1\}^* \\ abracadabra &\in \{a,b,\ldots,z\}^* \end{aligned}$$

Subwords, prefixes, suffixes

$w[i..j]$ denotes the subword of w starting at the i-th letter and ending at the j-th letter. If $i = 1$ this is a prefix, and if $j = |w|$ a suffix.

Lock puzzle

Say that we have a lock which a 3-digit code and the usual 0-9 keypad. The lock remembers the last 3 digits, and opens if they're the same as the secret code, i.e., there is no enter key.

Example

123 after pressing 7 becomes 237.

Question

How many times do you need to press a button to check all possible codes?

For 0-1 keypad, a good choice would be 0001011100.

Lets try to formulate the question in the language of words!

Lock puzzle, general version

For an alphabet Σ and an integer k, we want to find the shortest possible word w such that **any** word of length k over Σ occurs at least once as a subword in w.

There are two question:

1. what is the smallest possible length?
2. can we actually construct the corresponding w quckily?

Whenever faced with a task like this, you should first try to find some lowerbound on the solution size. In some ☺ cases a reasonable lowerbound will be actually the real answer.

How many words of length k over Σ there are? $|\Sigma|^n$. Hence the length of our solution must be at least $|\Sigma|^k + k - 1$.

Lemma

There exists a word of length $|\Sigma|^k + k - 1$ which contains each word of length k over Σ exactly once as a subword.

We will try to prove this lemma. The key insight is to reformulate the whole question in the language of graphs.

First approach

Nodes of the graph should correspond to possible words of length k. What should be the edges? $u \xrightarrow{c} v$ if u becomes v after pressing key c.

Nice try, but not good enough.

Maybe words should correspond to edges, not nodes?

Second approach

Nodes of the graph should correspond to possible words of length $k-1$. What should be the edges? $u \xrightarrow{c} v$ if $v = u[2..k-1]c$.

Now, any solution w corresponds to a walk of length $|w|-(k-1)$ which visits each edge at least once.

Any walk of length ℓ which visits each edge at least once corresponds to a solution w of length $\ell+(k-1)$.

So we can completely forget about the original question, and focus on finding shortest such walk in the graph $G_{\Sigma,k}$.

Now, look at the graph again. What can be said about the out- and in-degrees of the vertices?

Oservation

For any Σ and k, the graph $G_{\Sigma,k}$ is eulerian.

Hence we can actually find a walk which visits each edge exactly once. This is clearly the best possible, and as the number of edges in the graph is Σ^k, proves the lemma.

Lemma (what we proved)

There exists a word of length $|\Sigma|^k + k - 1$ which contains each word of length k over Σ exactly once as a subword.

We want to be computer scientists, so in most of the cases $\Sigma = \{0, 1\}$. Nevertheless, sometimes we need to work with a larger alphabet. A natural solution is to encode it using zeroes and ones.

$\Sigma = \{a, b, c, d\}$

$$\begin{aligned} a &\rightarrow 00 \\ b &\rightarrow 01 \\ c &\rightarrow 10 \\ d &\rightarrow 11 \end{aligned}$$

Now consider the following scenario: we have a very long string $w \in \Sigma$, which we need to encode in binary. How should we encode single characters?

$\Sigma = \{a, b, c, d\}$ and $w = a^{10000}bcd$

$$\begin{array}{ccc} a & \rightarrow & ? \\ b & \rightarrow & ? \\ c & \rightarrow & ? \\ d & \rightarrow & ? \end{array}$$

We should first formulate the question in more precise terms.

Alphabet encoding, general version

Given the frequencies of all characters of an alphabet Σ (i.e., $f(c)$ is the number of times c occurs), find a mapping $h : \Sigma \rightarrow \{0, 1\}^*$ such that:

1. no $h(c)$ is a prefix of $h(c')$, if $c \neq c'$,
2. the sum $\sum_{c \in \Sigma} |h(c)| f(c)$ is minimized.

$\Sigma = \{a, b, c, d\}$ and $w = a^{10000}bcd$

$$\begin{array}{lcl} a & \to & 0 \\ b & \to & 10 \\ c & \to & 110 \\ d & \to & 111 \end{array}$$

We should first formulate the question in more precise terms.

Alphabet encoding, general version

Given the frequencies of all characters of an alphabet Σ (i.e., $f(c)$ is the number of times c occurs), find a mapping $h : \Sigma \to \{0, 1\}^*$ such that:

1. no $h(c)$ is a prefix of $h(c')$, if $c \neq c'$,
2. the sum $\sum_{c\in\Sigma} |h(c)|f(c)$ is minimized.

This time we will focus on how to construct the best possible encoding in a reasonable time instead of looking for a simple formula.

How to visualize the solution?

We think that the encoding is a binary tree (or, more precisely, a trie). The root corresponds to the empty word ϵ, and each node to a word spelled out when following the path from the root. Each $h(c)$ corresponds to a leaf.

A few simple observations:

1. each inner node has exactly two children,
2. each leaf corresponds to some $c \in \Sigma$,
3. we want to minimize

$$\sum_i f(c_i)\,\text{depth}(v_i)$$

where v_i is the leaf corresponding to the i-th character c_i, and depth(v) is the depth of v.

The task looks hopeless. We could try all possible trees, but for $|\Sigma| = 256$ this would never finish before the end of the Universe.

We will try to develop a greedy method for this problem. The key insight will be that we can (quickly) reduce the problem P to a smaller one P' so that given an optimal solution to P' we can (quickly) recover an optimal solution to P.

Let c_i be the least frequent character. What can we say about the depth of v_i in an optimal tree?

Let c_i and c_j be the two least frequent characters (with $i \neq j$). What can you say about the depths of v_i and v_j in an optimal tree?

Lemma

Let c_i and c_j be the two least frequent characters. Then there is some optimal solution where v_i and v_j are brothers.

Proof.

Take any optimal solution (i.e., an optimal tree), and choose any inner node v whose both children are leaves maximizing depth(v). Then:

1. if both v_i and v_j are children of v, we are done,
2. if none of v_i and v_j are children of v, we swap them with the two children of v,
3. if v_i is a child of v but v_j is not, we swap v_j with the other child of v,
4. if v_j is a child of v but v_i is not... symmetry.

□

This lemma actually gives us an efficient way of solving the original problem.

Efficient?

Computer scientists usually denote the size of the problem by n. In our case, $n = |\Sigma|$. Then they are happy if the number of steps required to implement their method is roughly polynomial in n.

Roughly?

To make the notion of roughly more precise, we use the big-oh notion:

$$f(n) \in O(g(n)) \iff \exists_{\alpha>0}\exists_{n_0}\forall_{n>n_0} f(n) \leq \alpha g(n)$$

Now, using the lemma we get the following simple method:

1. if $\Sigma = 2$, return the trivial tree with one inner node
2. choose c_i and c_j with the smallest frequencies (and $i \neq j$)
3. "glue" characters c_i and c_j together, i.e., create a new character c with frequency $f(c_i) + f(c_j)$ and forget about c_i, c_j
4. construct an optimal solution recursively
5. replace in this optimal solution for the smaller problem the node corresponding to c with an inner node with two children corresponding to c_i and c_j

How many steps do we need to solve the problem of size n?

$O(n^2)$ naively. But, we can do much better!

Forget about actually constructing the solution. Say that we only want to know its value. Then the following recursive formulation can be changed so that we only need a data structure which stores the frequencies and allows:

1. inserting a new number,
2. extracting the smallest number.

It is known that using something called a heap we can implement both operations in $O(\log n)$ steps, hence the whole complexity becomes $O(n \log n)$.