Analysis of Algorithms

- An algorithm is a sequence of computational steps that transform the input into the output.
- It typically solves some problem.
- Computer programs are more general than algorithms.

Analysis

- Previously algorithms were implemented, and running time was measured.
- There are difficulties in this approach: different languages, different machine speed, and different optimization.

General Framework

Let P be a problem, and let I be an input (instance) to P. The time complexity of an algorithm for P is measured in terms of the size of I.

Unit cost model

- Each instruction requires one unit of time.
- Each register requires one unit of space.

e.g., sorting n numbers: $|I| = n$.

Logarithmic cost model

- Space : $\log n$ for integer n
- Time : proportional to the length of the operands ($\log n$)

e.g., prime number testing of an integer n: $|I| = \log n$.

Worst Case and Average Case Analysis

Worst case time complexity:

- longest running time of an algorithm for any input of size n.
- It is an upper bound of the time complexity of the algorithm.
- It is easier to analyze.
- For some algorithms, worst case occurs fairly often. Average case is often as bad as worst case.

Average case or expected running time:

It is not easy to identify "average" input. Often we assume that all inputs of a given size are equally likely. Then expected running time is the average of running time for all input of size n.

String Matching

Strings are defined on a set of characters called *alphabet*, e.g., $\Sigma = \{0, 1\}$ or $\{a, b, \ldots, z\}$.

A string is a sequence of characters in Σ, e.g., 0010, 1001, sequence, character.

Given text string $T[1..n]$ and pattern string $P[1..m]$ over Σ, where $n \geq m$, find all occurrences of P in T.

```
T: abcabaabcabac
P: abaa
```

- Applications: text editor (vi, emacs, ...), database search, etc.
- Variations: approximate string matching (spelling correction), multiple keyword search (library), etc.

Naive String Matching Algorithm

```
0  procedure Naive(T,P)
1    for i = 1 to n-m+1 do
2    occur = true
3      for k = 0 to m-1 do
4        if T[i+k] != P[1+k] then occur = false fi
5      od
6      if occur == true then print "occurrence at i" fi
7    od
8  end
```


Analysis

Assign a constant to each step, and add all steps.

$$\begin{aligned} T(n,m) &= c_0 + c_1(n-m+1) + c_2(n-m+1) + c_3 m(n-m+1) \\ &\quad + c_4 m(n-m+1) + c_5 m(n-m+1) + c_6(n-m+1) \\ &\quad + c_7(n-m+1) + c_8 \\ &= c_0 + c_8 + (c_1 + c_2 + c_6 + c_7)(n-m+1) \\ &\quad + (c_3 + c_4 + c_5) m(n-m+1) \\ &= amn - am^2 + bn + cm + d \end{aligned}$$

It is both worst case and average case time of procedure Naive.

$f(n,m) = O(g(n,m))$ if there exist positive constants c, n_0 and m_0 such that $0 \leq f(n,m) \leq cg(n,m)$ for all $n \geq n_0$ and $m \geq m_0$.

Thus the time complexity of Naive is $O(mn)$. That is, it is enough to count the number of comparisons in step 4.

Optimize Naive Algorithm

```
0  procedure Naive(T,P)
1    for i = 1 to n-m+1 do
3      for k = 0 to m-1 do
4        if T[i+k] != P[1+k] then go to 7 fi
5      od
6      print "occurrence at i"
7    od
8  end
```

Worst case: $O(mn)$

```
T: aaaaaaab     P: aaab
```

Best case: $O(n)$

```
T: aaaaaaaa     P: baaa
```


Average Case Time Complexity of Naive

Assume that input strings are random, i.e., all strings of a same size are equally likely. For example, if $\Sigma = \{a, b\}$ and $m = 4$, then the probability that `abaa` is a pattern is $1/16$.

Let $q = |\Sigma|$. For pattern, q^m strings are equally likely. For text, q^n strings are equally likely.

English is not like this, nor programming languages. But as far as string matching is concerned, it is a reasonable assumption.

The probability of a match (conditioning on text characters): There are q text characters with each probability $1/q$. For each text char, the probability that pattern character matches it is $1/q$. So the probability of a match is $q(1/q)^2 = 1/q$. Let $p = 1/q$.

- X_i: random variable that represents the number of comparisons for position i
- X: random variable that represents the total number of comparisons.

Then $X = X_1 + \cdots + X_{n-m+1}$. Thus

$$E[X] = E[X_1 + \cdots + X_{n-m+1}] = (n-m+1)E[X_i].$$

$$\begin{aligned} E[X_i] &= 1(1-p) + 2p(1-p) + \cdots + (m-1)p^{m-2}(1-p) + mp^{m-1} \\ &= (1-p)(1 + 2p + \cdots + (m-1)p^{m-2}) + mp^{m-1} \\ &= (1-p)\frac{1 - mp^{m-1} + (m-1)p^m}{(1-p)^2} + mp^{m-1} \\ &= \frac{1-p^m}{1-p}, \end{aligned}$$

where

$$\sum_{k=1}^{n} kx^{k-1} = \frac{1 - (n+1)x^n + nx^{n+1}}{(1-x)^2} \quad \text{for } x \neq 1.$$

Or, use formula $E[X] = \sum_{k=1}^{\infty} P\{X \geq i\}$ (see p320 of Ross):

$$\begin{aligned} E[X_i] &= 1 + p + p^2 + \cdots + p^{m-1} \\ &= \frac{1-p^m}{1-p}. \end{aligned}$$

Since $E[X_i] < \frac{1}{1-p} = \frac{q}{q-1} \leq 2$,

$$E[X] = (n-m+1)E[X_i] \leq 2(n-m+1).$$

That is, the expected running time of Naive is $O(n)$.