String Matching Z
(Following Gusfield Chapter 2)
Microscopy of chromosomes of a human female (karyotype):

Where does this string occur in the genome?
Exact String Matching

**Exact String Matching Problem.** Given a (long) string $T$ and a shorter string $P$, find all occurrences of $P$ in $T$. Occurrences of $P$ are allowed to overlap.

- Motivation is obvious:
  - search for words in long documents, webpages, etc.
  - find subsequences of DNA, proteins that are known to be important.

- We’ll see 4 efficient algorithms for this problem.
The Simple (Slow) Algorithm

SimpMatch(T, P):
   for i = 1..|T|:
       j = 1
       while j ≤ |P| and T[i+j-1] == P[j]:
           j += 1
       if j == |P|+1: print "Occurs at", i

- Runs in O(|T|×|P|) time.

- Information gathered in while loop at iteration i is ignored in iteration i+1.

- Key idea for speeding it up: use what we learned about T in the while loop to increment i by more than 1 in the outer loop.
Exploiting Patterns in $P$

- After comparing “happy” to “happe” at iteration $i$,
  - we know that $T[i...i+3] = \text{“happ”} = P[1...4]$
  - we can deduce that there can be no match at $i+1$ because $T[i+1] = P[2] = \text{“a”}$ but $P[1] = \text{“h”}$
  - in fact, since “h” does not appear in $T[i...i+3] = P[1...4]$, we could set $i = i + 4$

- Since $T$ will have matched some part of $P$, it is the similarities between various parts of $P$ that allow us to make these deductions.

⇒ Preprocess $P$ to find these similarities.
Z-Algorithm
**Fundamental Preprocessing**

**Def.** $Z_i(P)$ = the length of the longest substring of $P$ that starts at $i > 1$ and matches a prefix of $P$.

- $P = \text{"aardvark"}$: $Z_2 = 1$, $Z_6 = 1$
- $P = \text{"alfalfa"}$: $Z_4 = 4$
- $P = \text{"photophosphorescent"}$: $Z_6 = Z_{10} = 3$
String Search With $Z_i$

$$Z\text{Match}(T, P):$$
\[ S = P \text{@} T \]
Compute all $Z_i$ for $S$
\[ \text{return all } i - |P| - 1 \text{ such that } Z_i = |P| \]
\[ \text{(map indices of } S \text{ to indices of } T) \]

Why does this work?

- $Z_i(S) = |P|$ if and only if the string starting at $i$ in $S$ matches $P$.
- Running time is $O(|P| + |T| + Z_S)$, where $Z_S$ is the time to compute the $Z_i$ for $S$.
- **Next:** an $O(|P| + |T|)$ algorithm for computing the $Z_i$. 
Z-Boxes

**Def.** Z-box at $i$ is the substring starting at $i$ and continuing to $i+Z_i-1$. This is the substring that matches the prefix. There is no Z-box at $i$ if $Z_i = 0$.

- Algorithm for computing $Z_i$ will iteratively compute $Z_k$ given:
  - $Z_2...Z_{k-1}$, and
  - the boundaries $l, r$ of the rightmost Z-box found starting someplace in $2...k-1$. 
Z Algorithm

- Input: $Z_2...Z_{k-1}$, and the boundaries $l, r$ of the rightmost Z-box found starting someplace in $2...k-1$.
- Output: $Z_k$, and updated $l, r$

1. If $k > r$, explicitly compute $Z_k$ by comparing with prefix. If $Z_k > 0$: $l = k$ and $r = k + Z_k - 1$ (since this is a new farther right Z-box).

2. If $k \leq r$, this is the situation:

Two subcases:

- $Z_{k'} < \beta$: Set $Z_k = Z_{k'}$ and leave $l, r$ unchanged.
- $Z_{k'} \geq \beta$: Explicitly compare after $r$ to set $Z_k$. $l = k$, $r =$ point where comparison failed.
Analysis

• Correctness follows by induction and the arguments we made in the description of the algorithm.

• Runs in $O(|S|)$ time:
  - only match characters covered by a Z-box once, so there are $O(|S|)$ matches.
  - every iteration contains at most one mismatch, so there are $O(|S|)$ mismatches.

• Immediately gives an $O(|P| + |T|)$-time algorithm for string matching as described a few slides ago.
  - $O(|P| + |T|)$ is the best possible worst-case running time, since you might have to look at the whole input.
  - But better algorithms exist in practice that, for real instances, have expected sublinear runtime.