Lecture 2

Exact Matching

Question: *Given a pattern (query), does it exist in the text (reference)?*

Example:

1 i n
ATTCACTATTCGGCTAT
GCAT

Naïve Approach: Look at the first pattern symbol in the text and go from there.

Start with the first character of the text and the first character of the pattern and compare each character individually. If a mismatch occurs move the pattern one character and repeat the procedure. Terminate when the remaining characters in the text are less than the characters in the pattern since a match is impossible.

Algorithm:

```
for i=1,n-k
      for j=1,k
            if P[j] != T[i+j-1]
                  goto NOMATCH
      end for
      found match @ i
      NOMATCH
end for
```

Running time: Order of $O(n^k)$. Worst case $(n-k+1)^k = nk - k^2 + k$

Question: *When will the worst case occur?*

In a string where the entire pattern has to be compared for each character in the text. The pattern will repeatedly fail to match on the final character.

Example:

1 i n
AAAAAAAAAAAAAAAA
AAAAT (k comparisons)
AAAAT (k comparisons)
AAAAT (k comparisons)
... (repeat for n-k characters)

How can we improve on this? Can we use knowledge about comparison's we've already

made?
Pre-processing Approach: Information collected in the pre-processing step can speed up the subsequent matching process.

Z – Algorithm: Created by Gusfield in order to better explain other string matching algorithms.

Definition: Z[i]: length of longest common prefix of T[i..h] and T[0..n]

Text:	A T T C A C T A T T C G G C T A T
Z[i]	0 0 0 0 1 0 0 4 0 0 0 0 0 0 0 2 0

Question: *Can Z-values help in aligning a pattern to a text?*

In order to identify occurrences of the pattern in the text define:
S = P$T,

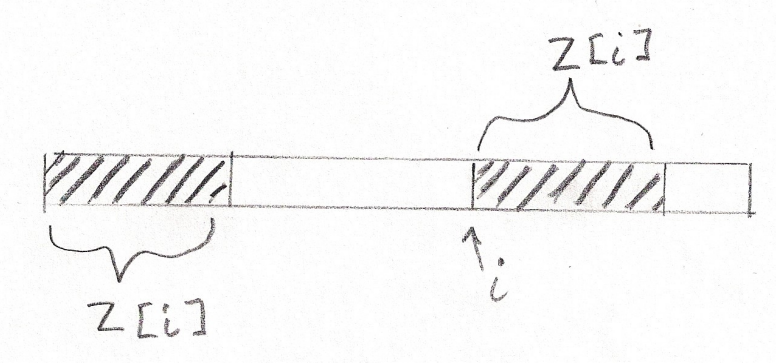
where $ is a symbol that doesn't occur in the pattern and in the text. An occurrence is detected if there is a Z[i] with i equal to the length of the pattern. By using this approach detecting the pattern is done by finding the prefixes of the string S.

This algorithm does matching in linear time and it does not depend on the alphabet that we are using. If every character we see is something we haven't seen before then we get the worst computation time!

Question: *Can Z-values be computer in linear time?*

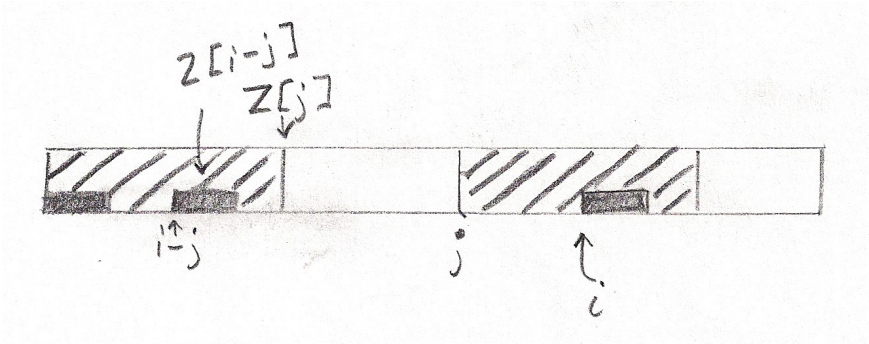
Overview of Z-Algorithm:

Z[i] will equal the number of characters starting at position i that match the prefix of the pattern.

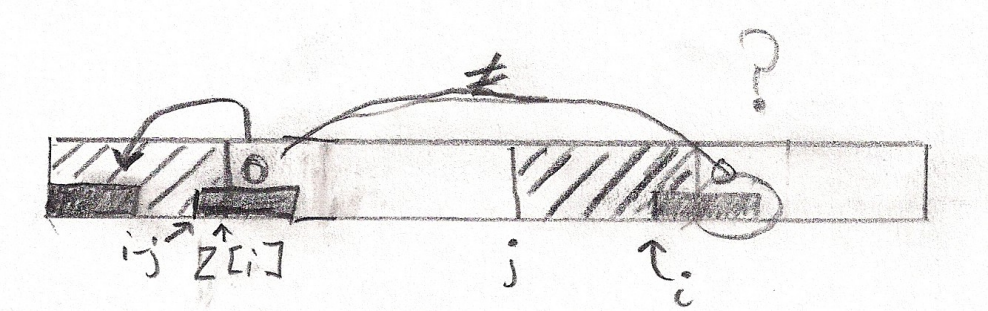


The idea is that we can use a previous Z value at position j to aid us in determining Z[i].

One example is if Z[j] extends past the matched portion position i (as diagramed below), then we already know the value of Z[i]. The characters starting at i must have already been matched to the start of the string. These character comparisons would have occurred when computing the Z value for i-j. Therefore, Z[i] = Z[i-j].



What if the matched portion of i extends past our previous furthest extending Z[j] *(i – j + Z[i-j] > Z[j])*? Then the substring that contributes to Z[i-j] goes beyond our current j + Z[j]. We can set Z[i] to Z[j] + j – i + match the prefix starting @ j + Z[j]

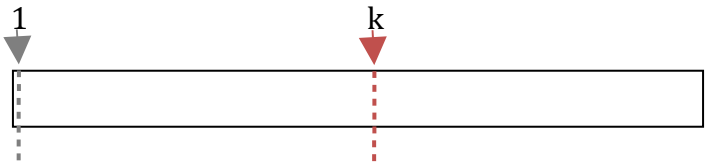


If a character matches, never look at it again. <=2m total character comparisons.

Z-Algorithm:

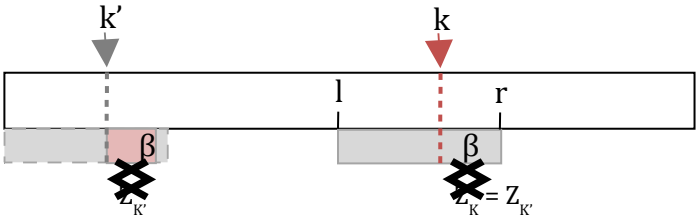
for i := 2,…,n either case 1 or case 2 applies:

1. if **i > r** then compare the characters starting at k with the characters starting at 1. If a mismatch occurs then set Z_K to the number of characters that matched.
   If $Z_i > 0$, set l := i and r := i + Z_i - 1; else l,r :=0

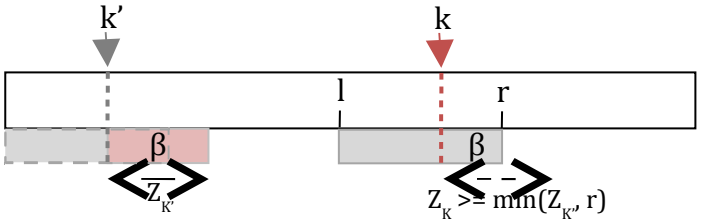


2. if **k ≤ r**, the position k is inside a Z-box S[l…r] = S[1…Z_l]. Thus S[k…r] = S[k'…Z_l].
   Let β = S[k…r];

   a) If $\mathbf{Z_{K'} < |\beta|}$, then the substring that contributes to ZK' lies inside the box and we can simply set $Z_k := Z_{k'}$.



b) If $\mathbf{Z_{K'} \geq |\beta|}$, then the substring that contributes to ZK' goes beyond the current Z-Box (>r) and we can set ZK to the right side of the current box (r) and explore the succeeding characters by individually comparing it with the prefix.

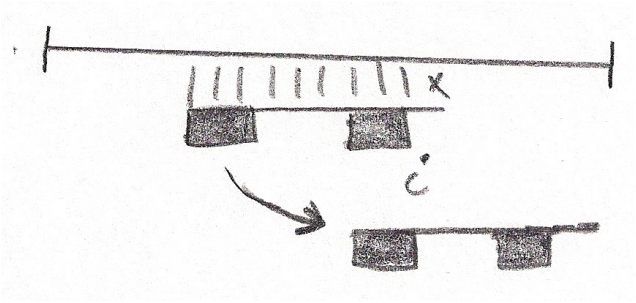


KMP – Knuth, Morris, Pratt Algorithm:

1 i n
XYABCXABCXADCDAFEA
..ABCXABCDE
→ABCXABCDE

SP[i] longest suffix of P from 1 … I that matches prefix of pattern.

Jump i – SP[i] characters on mismatch:



[End of lecture]

Algorithm:
For j =1 to n:

```
        i = j + Z[j]
        SP(i) = Z[j]
End
```

Problems with this approach? SP values are overwritten. Easy solution: go from n to 1.

Does this fix all the problems? No