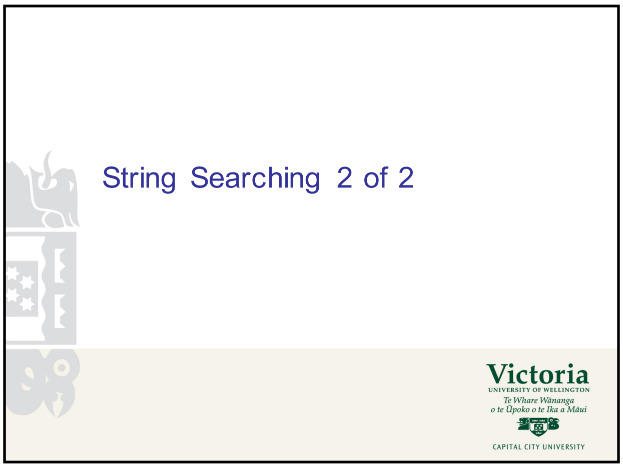
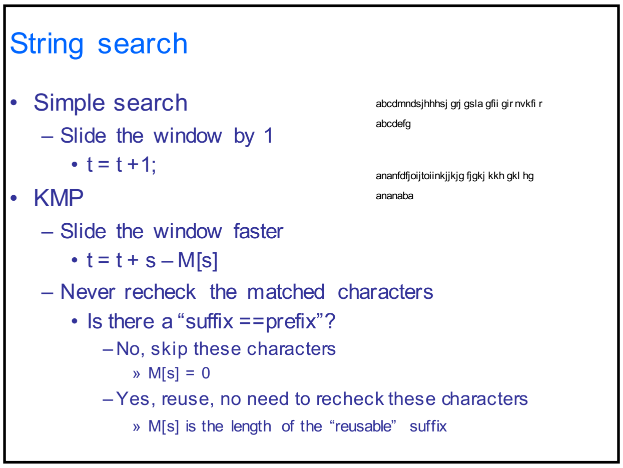


String Searching 2 of 2

Victoria
UNIVERSITY OF WELLINGTON
Te Whare Wānanga o te Ūpoko o te Ika a Māui
CAPITAL CITY UNIVERSITY



String search

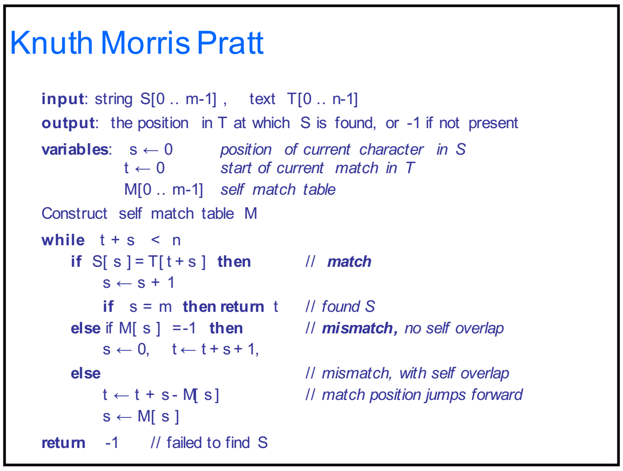
- Simple search
  - Slide the window by 1
    - t = t +1;
- KMP
  - Slide the window faster
    - t = t + s – M[s]
  - Never recheck the matched characters
    - Is there a “suffix ==prefix”?
      - No, skip these characters
        - » M[s] = 0
      - Yes, reuse, no need to recheck these characters
        - » M[s] is the length of the “reusable” suffix

abcdmndsjhhhsj grj gsla gfii gir nvkfi r

abcdefg

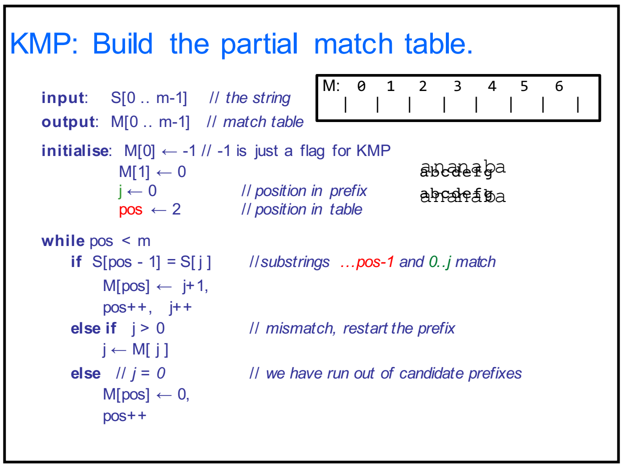
ananfdfjoijtoiinkjjkjg fjgkj kkh gkl hg

ananaba



Knuth Morris Pratt

```
input: string S[0 .. m-1] ,    text  T[0 .. n-1]
output:  the position  in T at which  S is found,  or -1 if not present
variables:  s ← 0          position  of current  character  in S
            t ← 0          start of current  match in T
            M[0 .. m-1]    self match table
Construct  self match table  M
while  t + s  <  n
    if  S[ s ] = T[ t + s ]  then          //  match
        s ← s + 1
        if  s = m  then return  t          // found S
    else if M[ s ]  =-1  then              // mismatch, no self overlap
        s ← 0,    t ← t + s + 1,
    else                                   // mismatch, with self overlap
        t ← t + s - M[ s ]                 // match position jumps forward
        s ← M[ s ]
return   -1     // failed to find S
```

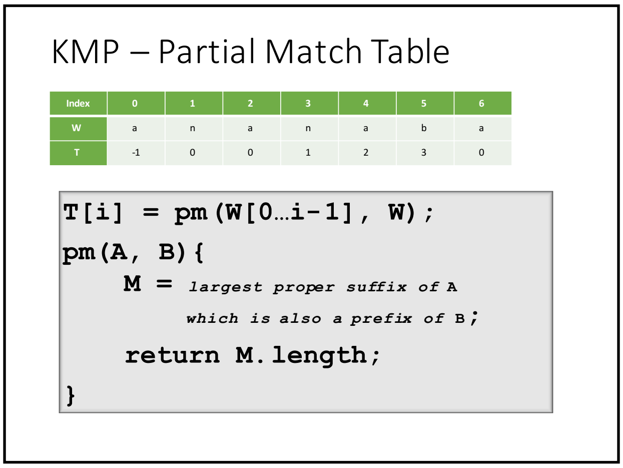


KMP: Build the partial match table.

```
M:  0   1   2   3   4   5   6
  |   |   |   |   |   |   |   |

input:    S[0 .. m-1]   // the string
output:   M[0 .. m-1]   // match table

initialise:  M[0] ← -1 // -1 is just a flag for KMP
             M[1] ← 0
             j ← 0            // position in prefix
             pos ← 2          // position in table

while pos < m
    if S[pos - 1] = S[ j ]     //substrings ...pos-1 and 0..j match
        M[pos] ← j+1,
        pos++,  j++
    else if  j > 0             // mismatch, restart the prefix
        j ← M[ j ]
    else  // j = 0             // we have run out of candidate prefixes
        M[pos] ← 0,
        pos++
```

ananaba
abcdefg

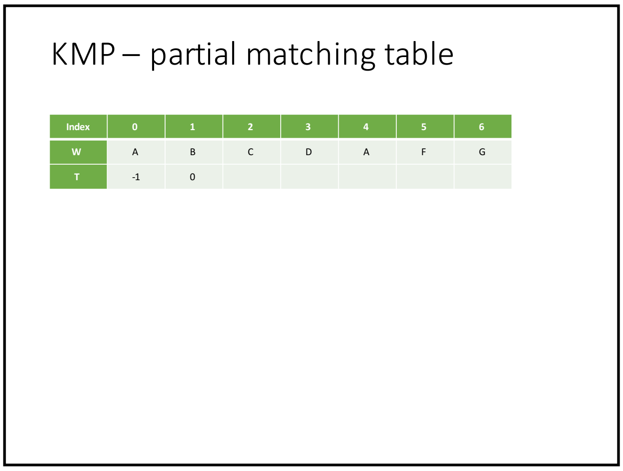
abcdefg
ananaba



KMP – Partial Match Table

Index	0	1	2	3	4	5	6
W	a	n	a	n	a	b	a
T	-1	0	0	1	2	3	0

```
T[i] = pm(W[0…i-1], W);
pm(A, B){
    M = largest proper suffix of A
        which is also a prefix of B;
    return M.length;
}
```



KMP – partial matching table

Index	0	1	2	3	4	5	6
W	A	B	C	D	A	F	G
T	-1	0					

KMP – example

Index	0	1	2	3	4	5	6
W	A	B	C	D	A	B	D
T	-1	0	0	0	0	1	2

ABCDABD
ABCABCDAABABABCDABCDABDE

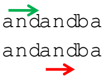
KMP – example

Index	0	1	2	3	4	5	6
W	A	A	A	A	A	A	A
T	-1	0					

KNP: Building the table.

```
M:  0   1   2   3   4   5   6   7
    |   |   |   |   |   |   |   |  .
```

```
input:    S[0 .. m-1]    // the string
output:   M[0 .. m-1]    // match table

initialise:  M[0] ← -1
             M[1] ← 0
             j ← 0              // position in prefix
             pos ← 2            // position in table

while pos < m
    if  S[pos - 1] = S[ j ]     // substrings ...pos-1 and 0..j match
        M[pos] ← j+1,
        pos++,  j++
    else if  j > 0              // mismatch, restart the prefix
        j ← M[ j ]
    else  // j = 0              // we have run out of candidate prefixes
        M[pos] ← 0,
        pos++
```



KMP – example (hard)

0	1	2	3	4	5	6	7	8	9	10
A	A	B	A	A	A	A	B	A	C	A

String search

- Knuth Morris Pratt
  - searches forward,
  - never matches a text character twice (and never skips a text character)
  - jumps string forward based on self match within the string:
    - prefix of string matching a later substring.
    - doesn't use the character in the text to determine the jump
  - Cost:
- Boyer Moore
  - Searches backward
  - Actually jump and skip many characters
  - Use the characters in the text to determine the jump

abanana
alongpieceoftextwithnofruit

Boyer Moore: string search

- string: s[0] .. s[m-1]
- text: t[0] .. t[n-1]

```
abanana
bananfanlbananabananafan
```

Why look at every character in the text?

Start searching from the end of the string, backwards

When there is a mismatch,

move the string forward by an appropriate jump and restart:

table 1: what was the text character that mismatched?

⇒ what is the shortest jump that could make a match?

table 2: what has already been matched

⇒ what is the shortest jump that would match again

(take the longer of the two jumps suggested)