Prof. Helmut Alt
Team 31. 20140174 Hyungrok Kim

Exercise 1

(a)

$\sum$ = {a, b, c}

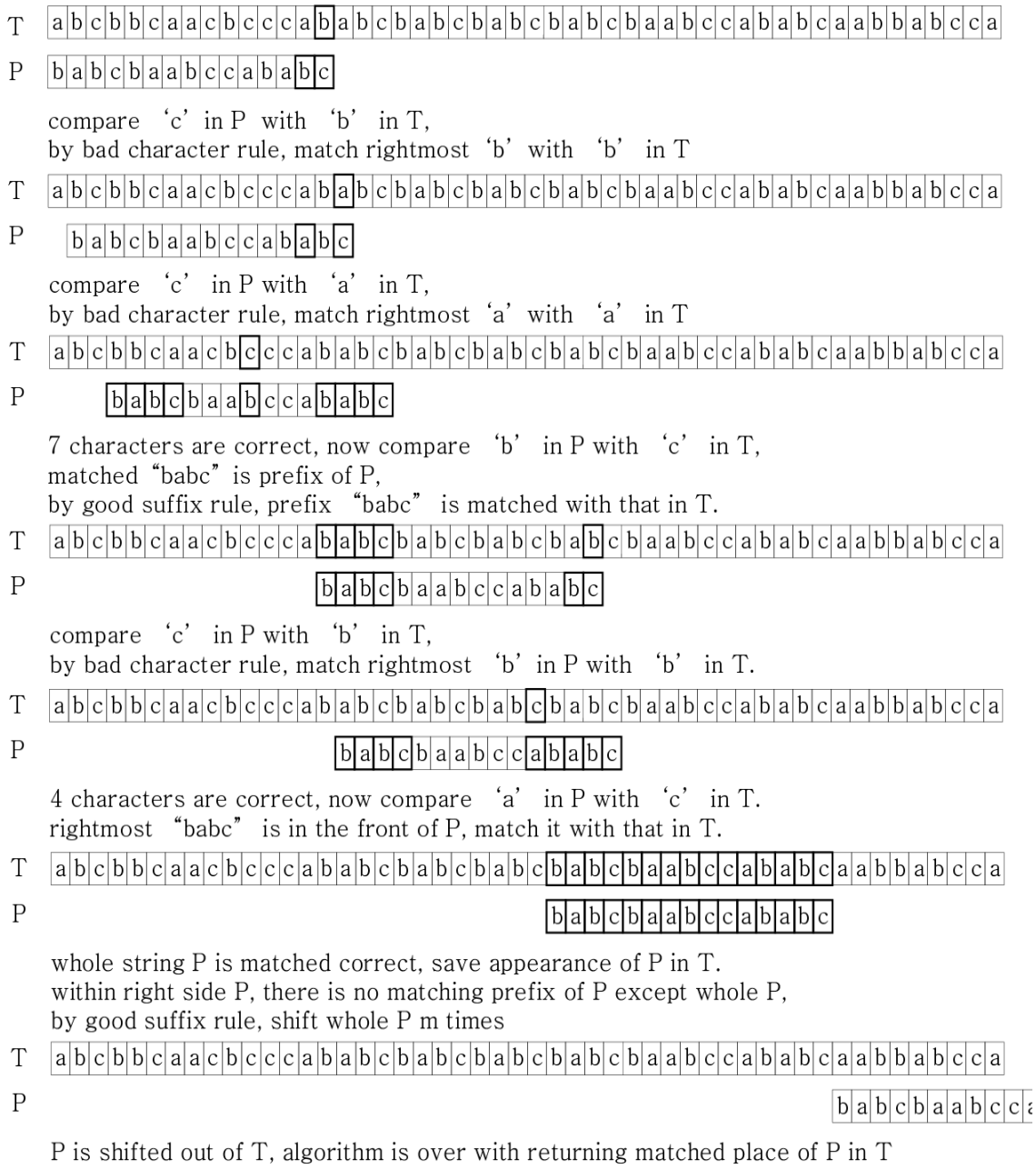
pattern P = babcbaabccababc

m = |P| = 15

text T = abcbbcaacbcccababcbabcbabcbabcbaabccababcaabbabcca

n = |T| = 50

If I visualize the procedure of Boyer-Moore algorithm with P and T,



(b)

pattern P

m = |P|

before determining array γ, we need to get the information that how the suffixes are appearing in the rest(previous) parts in pattern P.

let's store that in integer array with size of m, array suff[m]

in suff[i] ($0 \leq i < m$),

thinking of suffix starting from char P[m-1-i] to char P[m-1],

w=P[m-1-i]P[m-i]...P[m-1]

there're 2 cases,

case 1.

w is prefix of P then,

suff[i]=i+1 (which is |w|)

case 2.

w is not a prefix of P,

suffix of w is also suffix of P[0]P[1]...P[i]

w'(corresponding suffix of w),

$|w'|$=j, then suff[i]=j.

this algorithm to determine array suff is like,

```
void makesuff (char *P, int m)
    suff is linear integer array with size m.
    int f, g, i;
    suff[m-1]=m;   //because the suffix of size m is same with whole string P
    g=m-1;
    for i from (m-2) to 0,
        if   i>g and suff[i+m-1-f]<(i-g) //places at g(less then i) mismatch,
            then, suff[i] = suff[i+m-1-f]
        else
            if i<g //when last g(saved the unmatched location is not needed
                g=i
            f=i //save f as last i at next loop
            while g≥0 and x[g]==x[g+m-1-f]
                g-- //matching prefix and suffix of P from the back
            suff[i]=f-g
```

return array suff

now from the array suff

void make γ(char *P, int m)

γ is linear integer array of size m

int i, j

for i=0 to m-1,

γ[i]=m //initialize γ with m (which is max shifts)

j=0

for i from m-1 to 0, //checking from last index,

if suff[i]==(i+1) //case1

while j<m-1-i //restore array(where j is shift distance from original P)

if $\gamma[j]$==m, //which has m(old data which is renewable)

$\gamma[j]$=m-1-i

for i from 0 to m-2

//when case 2

$\gamma\left[m-1-suff[i]\right]=m-1-i$ //m-1-suff[i] is index in γ to make //condition of case 2, m-1-i is needed shifts

return array γ

from the algorithm makesuff and make γ,

we get proper array γ used in Boyer-Moore algorithm.

Now let's Analyze

for Space Complexity,

we need array suff and γ (both size m)

Space complexity = $O(m)$

for Time Complexity,

in makesuff,

operations with constant time is done in for loop(m times) except that while-loop which is O(m) independent to outer for-loop.

makesuff needs time complexity of O(m)

in make γ,

first-initializing for-loop need O(m),

in second for-loop, operation with O(1) and another for-loop is in it, but because integer j only increase with maximum O(m), so inner for-loop is indepently O(m)
last for-loop needs O(m)
make γ needs time complexity of O(m)
The time complexity for algorithm is O(m)