CX2101 Algorithm Design and Analysis

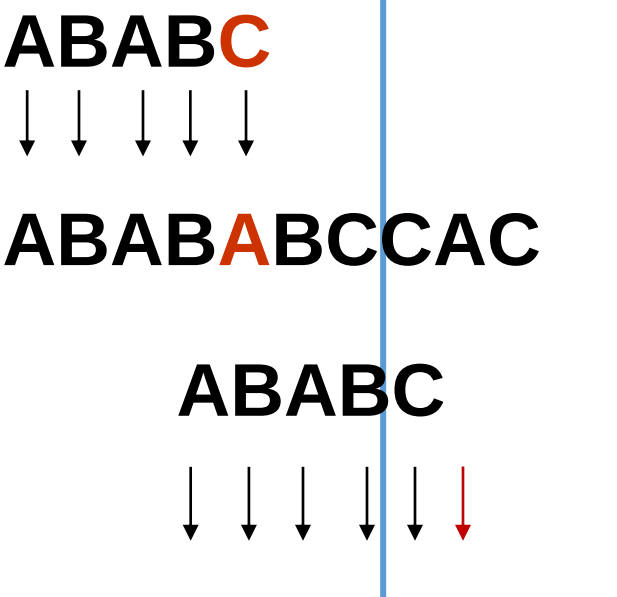
Tutorial 5
String Matching
(Week 12)

Question 1

Rewrite the simpleScan algorithm in the lecture slides to eliminate the variable i.

Original code

```
int SimpleScan (char [] P, char [] T, int m)
{
    int i, j, k;
    j = k = 0;
 i = 0;
 while (j <= n-m) {
     if (T[j] != P[k]) {
          j = ++i;
          k = 0; }
     else {
          j++;
          k++;
          if (k == m)   return i;   }
 return -1;
}
```



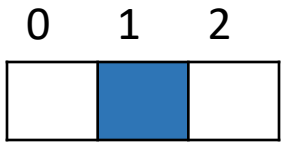
Question 1

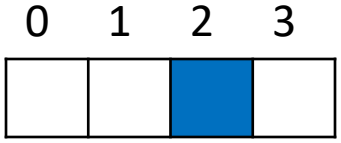
After removing i

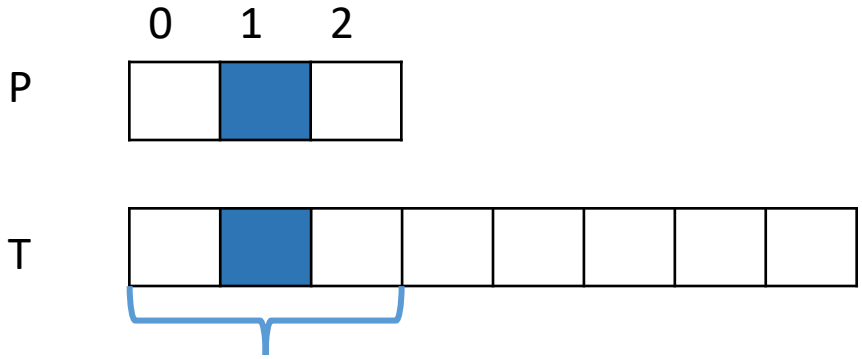
```
int SimpleScan (char [] P, char [] T, int m)
{
    int ~~i,~~ j, k;
    j = k = 0;
 ~~i = 0;~~
 while (j <= n-m) {
      if (T[j] != P[k]) {
            j = ~~++i;~~ j-k+1;
            k = 0; }
      else {
             j++;
            k++;
            if (k == m)    return ~~i;~~ j-k;   }
 return -1;
}
```

Question 2

How would you modify the Rabin-Karp algorithm to search for a given pattern with the additional condition that the middle character is a “wild card” (any text character at all can match it)?





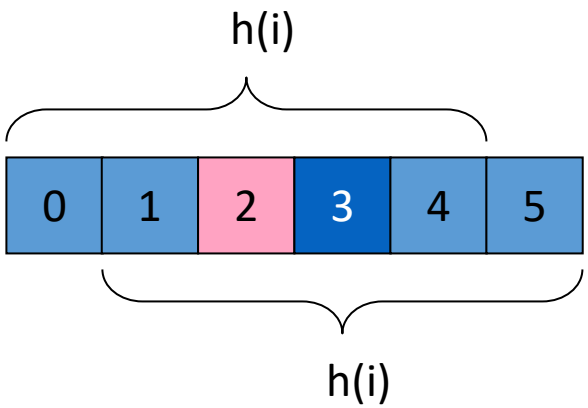


The hash function:
Replace the middle digit by 0

```
int hash(Txt, m, d)
{
        int h = Txt[0] % q;
        int middle = m/2;
        for (int i =  1; i < m; i++)
            if (i == middle)
                h = (h * d) % q;
            else
                h = (h * d + Txt[i] ) % q;
        return h;
}
```

To rehash after moving the text window one character right:

- Remove the MSB
- Add the middle digit in the old window
- Replace the middle digit in the new window by 0



In general,

$new = (old - MSB * d^{m-1} + MiddleB * d^{m/2}) * d - NewMiddleB * d^{m/2} + LSB$

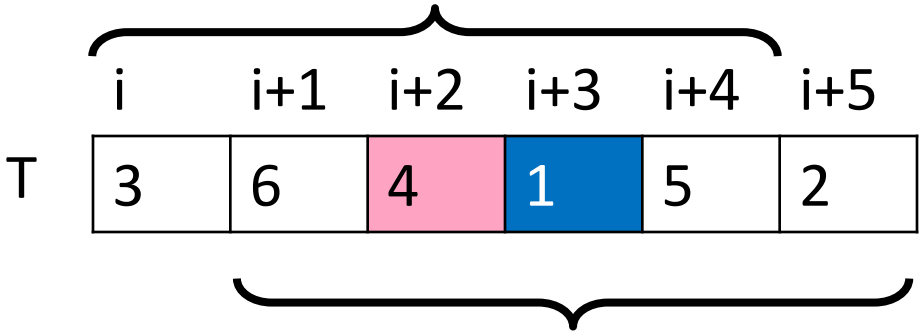
```
// dM = d^(m-1) % q
dM = 1;
For j = 1 to m-1
        dM = dM*d % q
```

```
// dMiddle = d^(m/2) % q
dMiddle = 1;
For j = 1 to m/2
    dMiddle = dMiddle*d % q;
```

```
int rehash(T, i, m, ht)
{     msb = (T[i] * dM) % q;   // dM = d^(m-1)
          oldMiddle = (T[i+m/2] * dMiddle) % q;
          newMiddle = (T[i+1+m/2] * dMiddle) % q;
          old_removed = ((ht + q) – msb + oldMiddle) % q;

     return (oldest_removed * d – newMiddle + T[i+m])
% q;
}
```

ht=0, dM = 4, dMiddle = 2, q = 7



Rehash(T, i, 5, 0) =

msb = 3*4 % 7 = 5
oldMiddle= 4*2 % 7 = 1
newMiddle = 1*2 % 7 = 2
oldRemoved = (0+7-5+1) %7=3
Return (30-2+2) % 7 = 2

Question 3

Given pattern P = “AAA……AB” (m-1 A’s followed by one B and text string T = “AAA…..A” (n A’s)

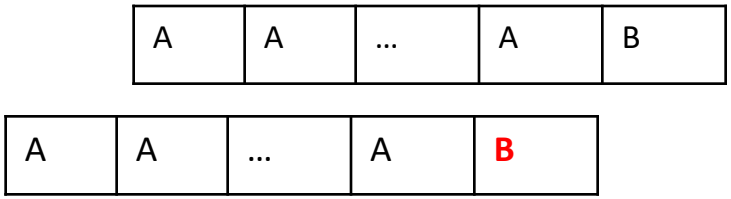
1) Show the values of CharJump and matchJump arrays for P computed by the Boyer-Moore string matching algorithm. Assume that alpahbet is {A,B,…,Z}.

2) Find out exactly how many character comparisons are done by simpleBMScan and BMScan respectively to scan T for an occurrence of P.

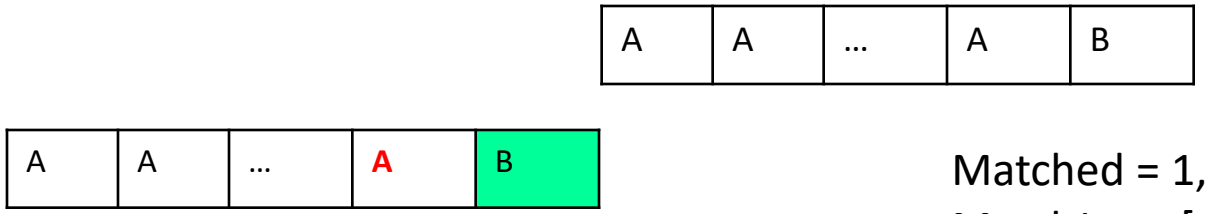
3(1)

P = “AAA……AB”

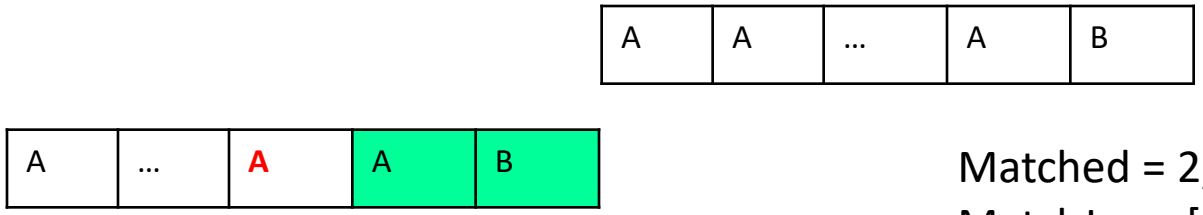
CharJump[‘A’] = 1
CharJump[‘B’] = 0
CharJump[x] = m



Matched = 0, Slide[m] = 1
MatchJump[m] = 1

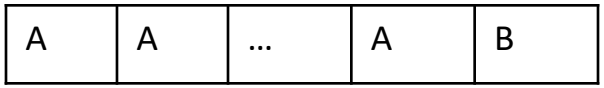


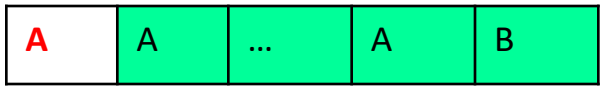
Matched = 1, Slide[m-1] = m
MatchJump[m-1] = m+1



Matched = 2, Slide[m-2] = m
MatchJump[m-2] = m+2

Matched = m-2, Slide[2] = m
MatchJump[2] = 2m-2





Matched = m-1, Slide[1] = m
MatchJump[1] = 2m-1

	1	2	…	m-2	m-1	m
MatchJump	2m-1	2m-2	…	m+2	m+1	1

3(2) simpleBMScan

j += **max(charJump[T[j]], m-k+1)**;

CharJump[‘A’] = 1, CharJump[‘B’] = 0, CharJump[x] = m

A A A……B

A A A……A……A 1st comparison then j+= max(1, m-m+1)=1

A A A….B

A A A……A……A 2nd comparison, j += 1

A A A……B

A A A……A……A (n-m+1)th comparison (the last)

- Therefore :

  No. of Comparisons = n – m + 1

3(2) BMScan

j += max(charJump[T[j]], matchJump[k]);

CharJump[‘A’] = 1, CharJump[‘B’] = 0, CharJump[x] = m

	1	2	…	m-2	m-1	m
MatchJump	2m-1	2m-2	…	m+2	m+1	1

A A A……B

A A A……A……A 1st comparison then j+= max(1, 1)

A A A……B

A A A……A……A 2nd comparison, j += 1

A A A……B

A A A……A……A (n-m+1)th comparison (the last)

- Therefore :

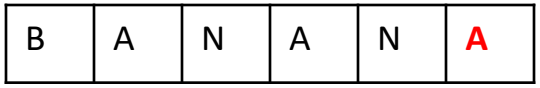
  No. of Comparisons = n – m + 1, the same as simpleBMScan

Question 4(1)

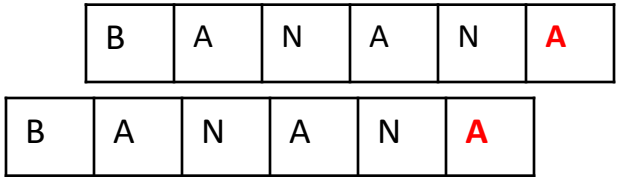
Show the values of CharJump and matchJump arrays for the following pattern, which are computed by the Boyer-Moore string matching algorithm, assuming alpahbet is {A,B,…,Z}.

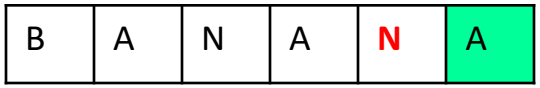
P = “BANANA”

CharJump[‘A’] = 0, CharJump[‘B’] = 5, CharJump[‘N’] = 1, CharJump[x] = 6

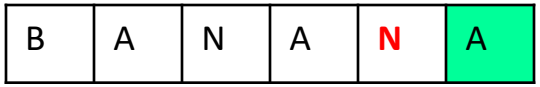


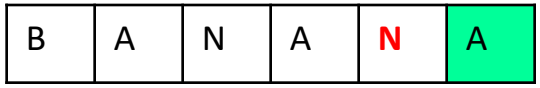
Matched = 0, Slide[6] = 1, MatchJump[6] = 1



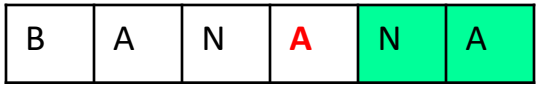


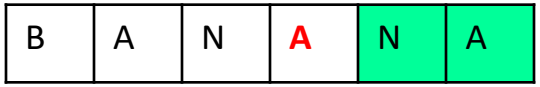
Matched = 1, Slide[5] = 4, MatchJump[5] = 5



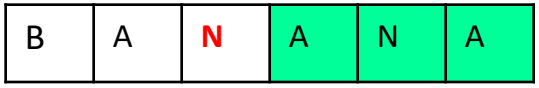


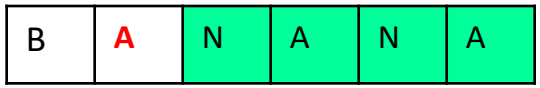
Matched = 2, Slide[4] = 6, MatchJump[4] = 8



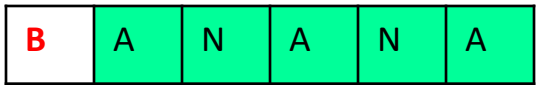


Matched = 3, Slide[3] = 2, MatchJump[3] = 5

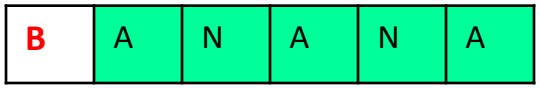


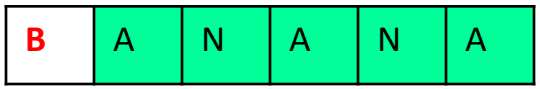


Matched = 4, Slide[2] = 6, MatchJump[2] = 10



Matched = 5, Slide[1] = 6, MatchJump[1] = 11



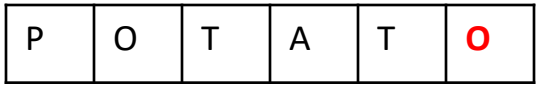


Question 4(2) – not covered if no time

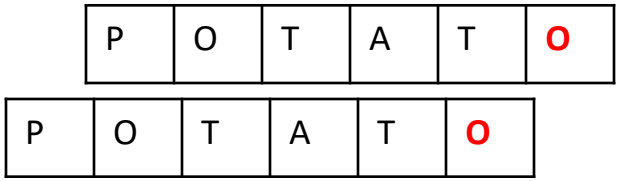
Show the values of CharJump and matchJump arrays for the following pattern, which are computed by the Boyer-Moore string matching algorithm, assuming alpahbet is {A,B,…,Z}.

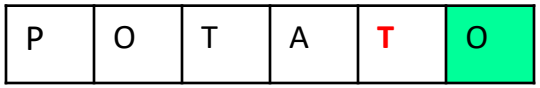
P = “POTATO”

CharJump[‘A’] = 2, CharJump[‘O’] = 0, CharJump[‘P’] = 5, CharJump[‘T’] = 1
CharJump[x] = 6

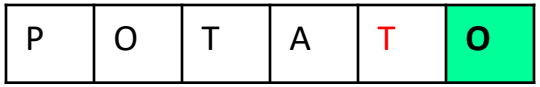


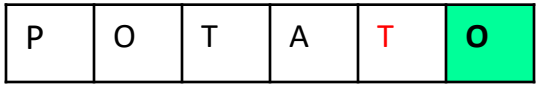
Matched = 0, Slide[6] = 1, MatchJump[6] = 1

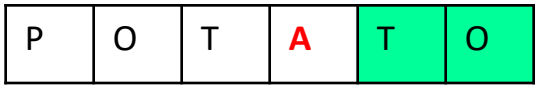




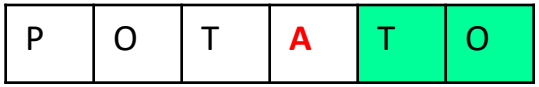
Matched = 1, Slide[5] = 4, MatchJump[5] = 5

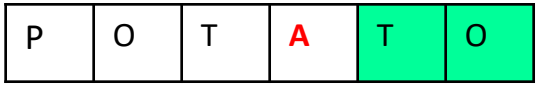


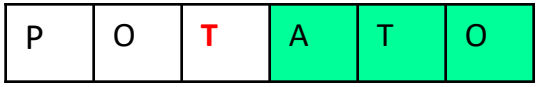




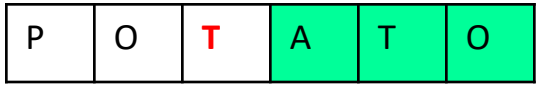
Matched = 2, Slide[4] = 6, MatchJump[4] = 8

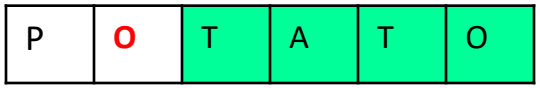




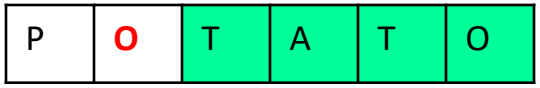


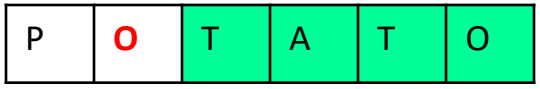
Matched = 3, Slide[3] = 6, MatchJump[3] = 9

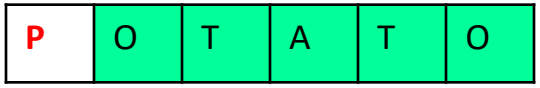




Matched = 4, Slide[2] = 6, MatchJump[2] = 10







Matched = 5, Slide[1] = 6, MatchJump[1] = 11