Course: Pattern Matching Algorithms(600.663) Lecture 3,4

Submitted by: Abhsihek Shreevats Date: 09/29/2010

One Dimension Pattern Matching

In the previous lecture, we learnt about some string matching algorithms like Knuth-Morris-Pratt, Aho-Corasick, Boyer-Moore and Bird-Baker. In this lecture, let us see a new method of looking for a pattern in text using a witness table. First we will examine the algorithm in 1 dimension and then we will expand the concept to 2 dimensions.

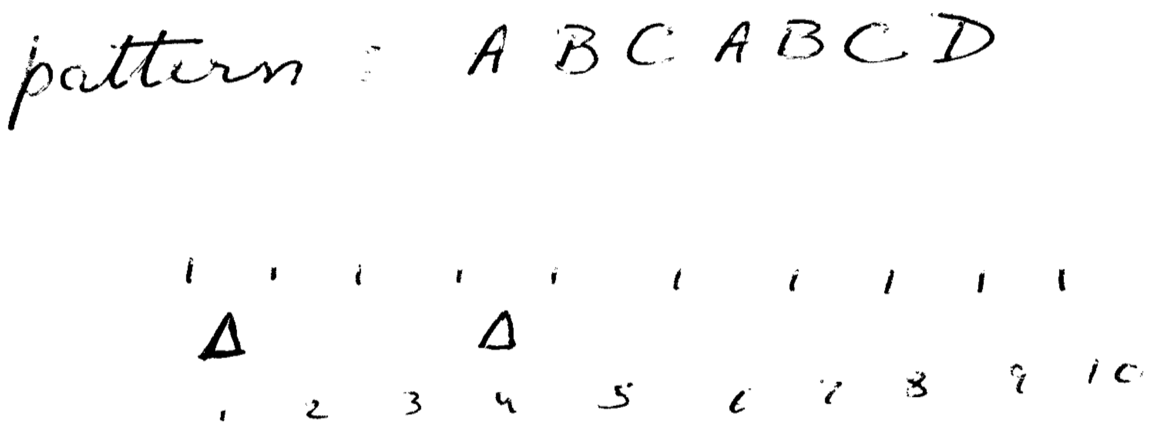

Figure 1

Consider figure 1 where the two marked locations are the candidates where a pattern may occur in the given text. The naïve way to verify if the candidates are true is to check for both the location character wise. Is there a better way? Is it possible to select a text location whose letter can eliminate one of the two candidates? By the look of it, the obvious answer seems to be the location 7. If the location 7 has a 'D' the second candidate is false and if not, the first candidate is false for sure. This is because, as shown in figure 2, the overlap between pattern starting at location 1 and that starting at 4 disagree at a certain point – 7th position from the start of the pattern.

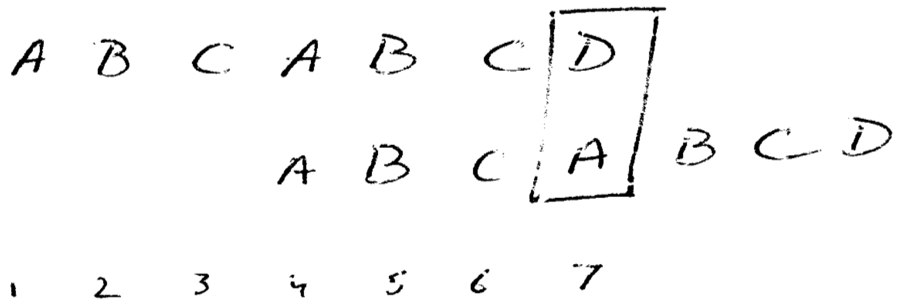

Figure 2

To put it in different words, for an overlap starting at 4, position 7 is a witness that decides which of the two placements is false. This process is called a *duel* and is valid only if the candidates are placed less than *m* distance away, as otherwise there will be no overlap.

Let us now learn how to build this witness table. To start with, we have a vector called *witness table* equal to the length of the pattern. For any position *i* in this vector, we put * if the pattern overlaps itself from this position without a mismatch and if there is a mismatch, we put the location where the mismatch occurs.

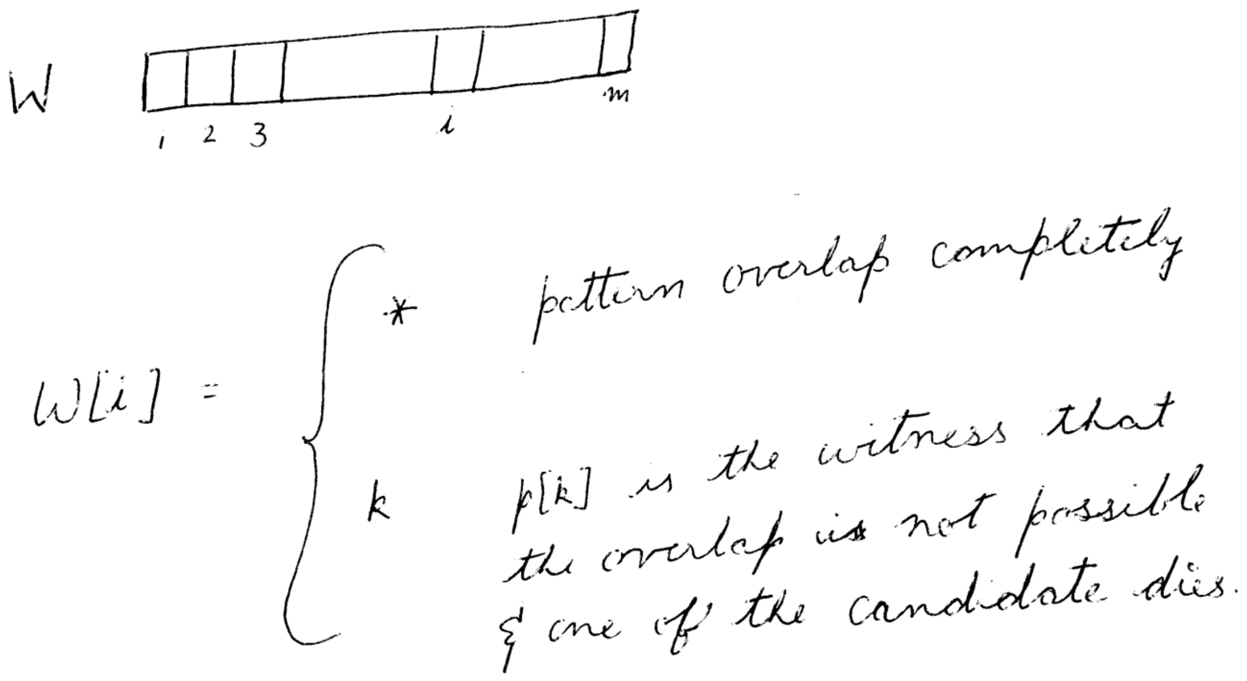
W
1 2 3
i
m
W[i] =
*
pattern overlap completly
k
p[k] is the witness that the overlap is not possible & one of the candidate dies.

Figure 3

To understand things better, let us build a witness table for the pattern AABBAABB.

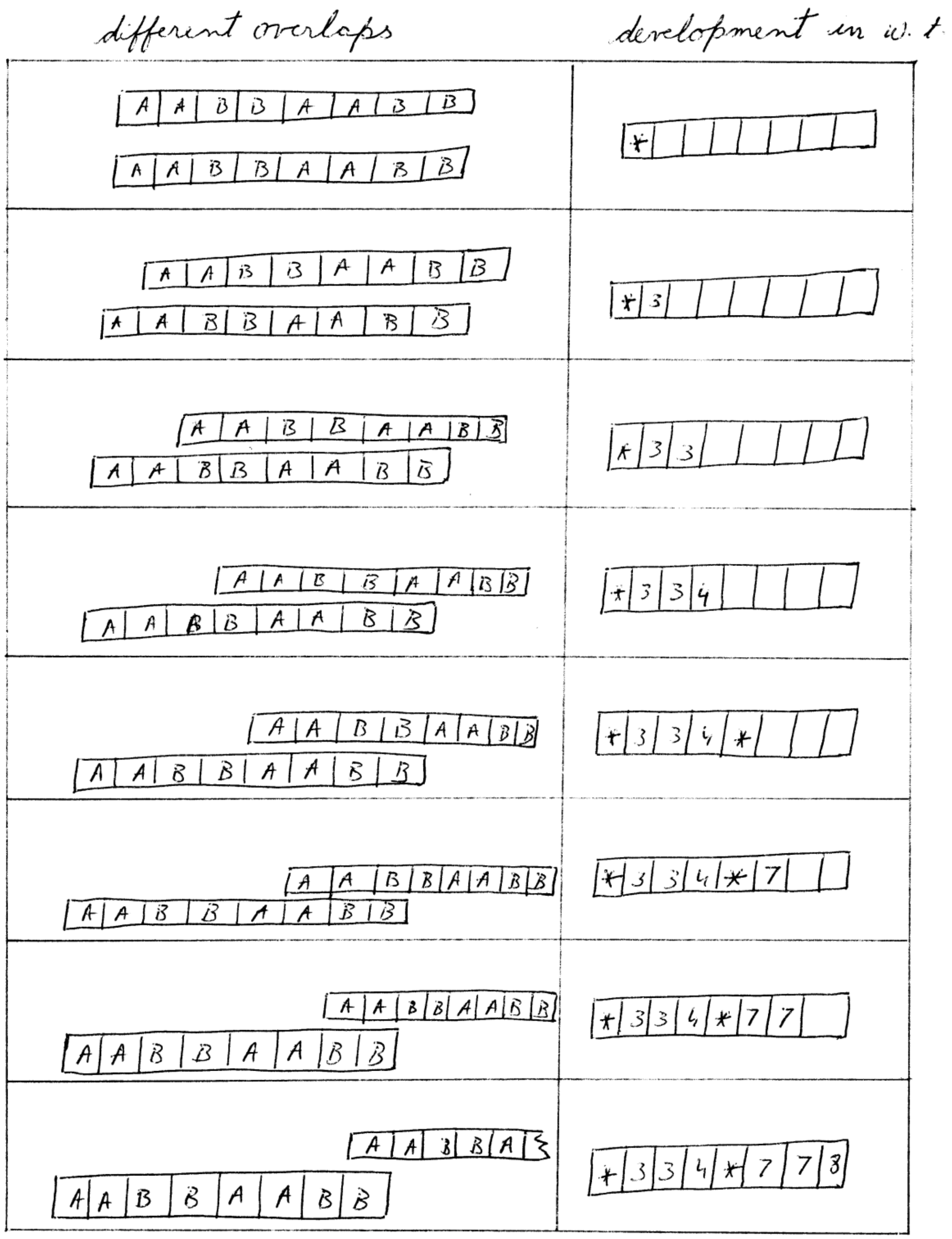
different overlaps
development in w.t
A A B B A A B B
*
* 3
* 3 3
* 3 3 4
* 3 3 4 *
* 3 3 4 * 7
* 3 3 4 * 7 7
* 3 3 4 * 7 7 8

Figure 4

Let us now see how do we use this witness table to resolve the duel between two candidates at location i and j, given i<j.

Algorithm:

```
if( j - i > m )
    no duel
else
    look at W[ j − i + 1 ]
    if( W[ j − i + 1 ] == k )
        if( T[ i + k - 1 ]  == P[k])
            kill j
        else
            kill i
    else //W[ j − i + 1 ] == *
        there is an agreement and no one dies
```

We look for a pattern in the text using dueling method following these two steps:

1. Eliminate all conflicting sources in a way that the remaining candidates are pair wise consistent(they overlap completely)
2. Verify all remaining candidates.

Let us look into each of these points in more details.

Eliminating conflicting sources

We start at the first 2 locations in the text. Duel will be between last potential source (a contender for a location where the pattern might start in the text) and the new untested location. Assume that the right most live candidate (live candidate: candidates which have survived duel) is R_c (refer figure 5). We will maintain a list of live candidates and assume that the new candidate is *New*.

Figure 5 gives a pictographic overview of this process.

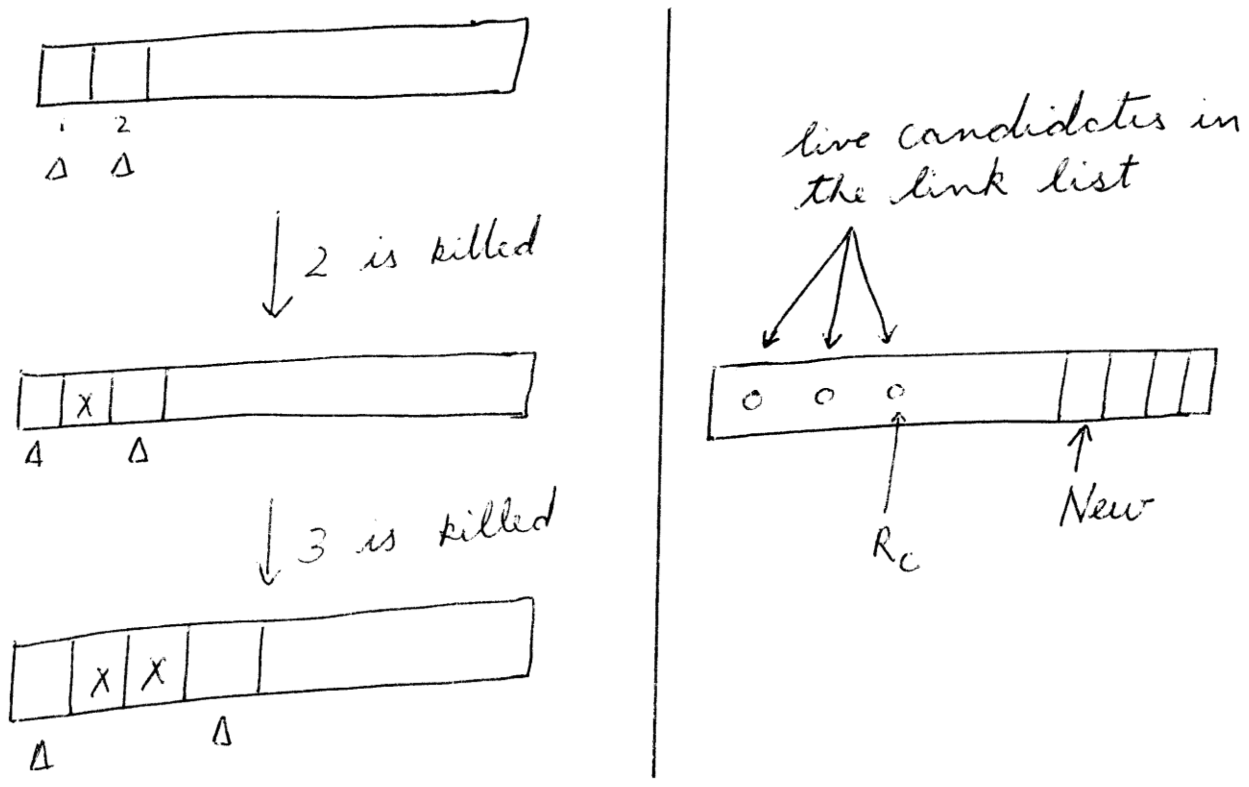

Figure 5

<u>Step wise algorithm for text scan to eliminate conflicting sources:</u>

Initial stage

R_c -> 1

New -> 1

Duel between R_c and New

if New – $R_c > m$ && length of text – New $\geq m$

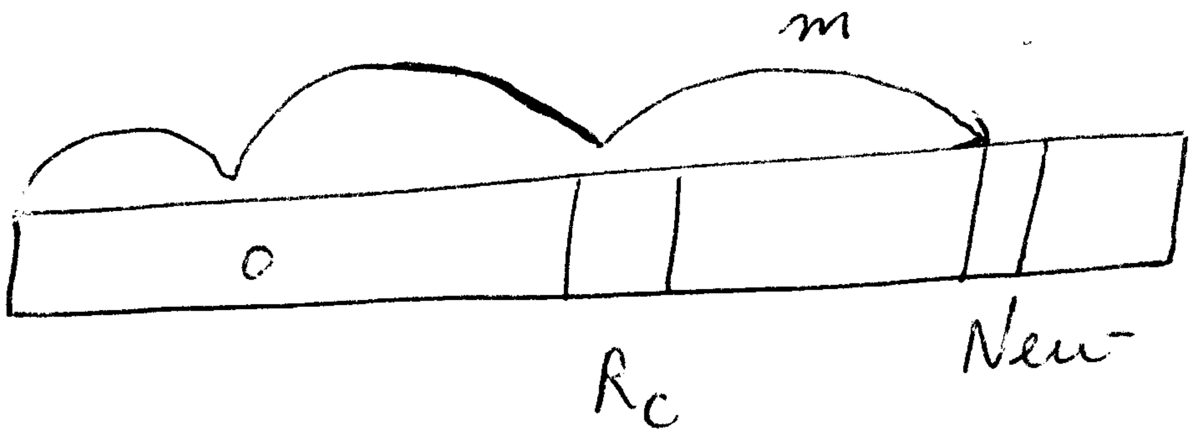

Figure 6

then,

add New to the list

R_c <- New

New <- New +1

if New - $R_c < m$

Cases arising from duel between New and R_c:

1. New dies

    R_c – same

    New <- New + 1

    Back to the head of the loop

2. R_c dies

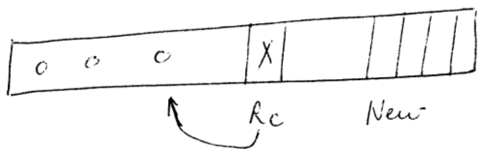

Figure 7

take R_c out of the list

R_c points to the previous element in the linked list

This continues till either New dies or New – R_c > m.

3. If there is a ∗ in the witness table at *New – R_c + 1*

```
add New to the list
R_c <- New
New <- New + 1
Back to head of the loop
```

Time

Elimination of conflicting sources during text scan can be done I linear time. Since every time *New* dies, we move in the forward direction only, therefore the contribution of this case is O(n). But what happens when R_c dies? Let us assume like in the Wild West, in a duel, the candidate who dies pays.

When R_c dies, we will charge the cost of going back to the previous candidate in the list on R_c. Since in this way, a candidate dies only once, the total cost cannot exceed O(n).

Therefore the cost of scanning the text to eliminate false candidate is O(n).

Correctness

So far we have not discussed if this elimination technique is correct.

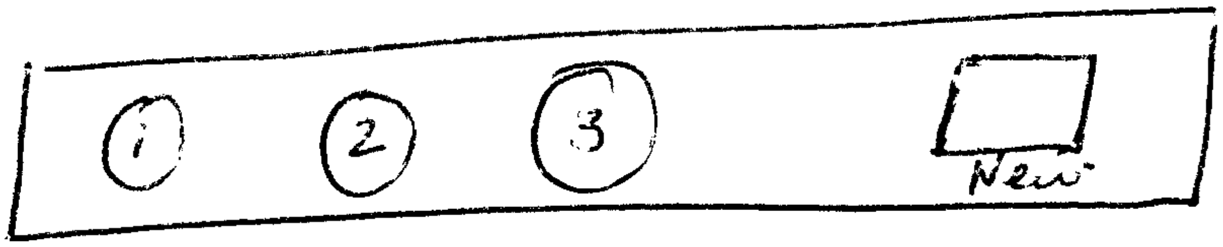

Figure 8

The following cases arise in the algorithm:

1. New dies – there seems to be no problem as we have a witness(witness table) for the mismatch.
2. $\boxed{3}$ dies – if $\boxed{3}$ dies we will duel between $\boxed{2}$ & New. Seems fine.
3. New and $\boxed{3}$ agree - a question that arise here is why did we not duel between $\boxed{2}$ and New in this case because we did so when $\boxed{3}$ died. Isn't it possible that $\boxed{2}$ and New have a conflict?

Let us first see why we had to duel between *New* and $\boxed{2}$ when $\boxed{3}$ died (please refer to figure 9).

$\boxed{3}$ died because of a conflict at location B. Here it is also possible that *New* and $\boxed{2}$ conflict at location A. Therefore we need to duel between $\boxed{2}$ and *New*.

However in the case where $\boxed{3}$ agrees with *New*, we need not duel between *New* and $\boxed{2}$ because of transitivity of equality i.e. we know $\boxed{2}$ and $\boxed{3}$ agree with each other, so basically, the overlap between *New* and $\boxed{2}$, being a subset of overlap between *New* and $\boxed{3}$ overlaps.

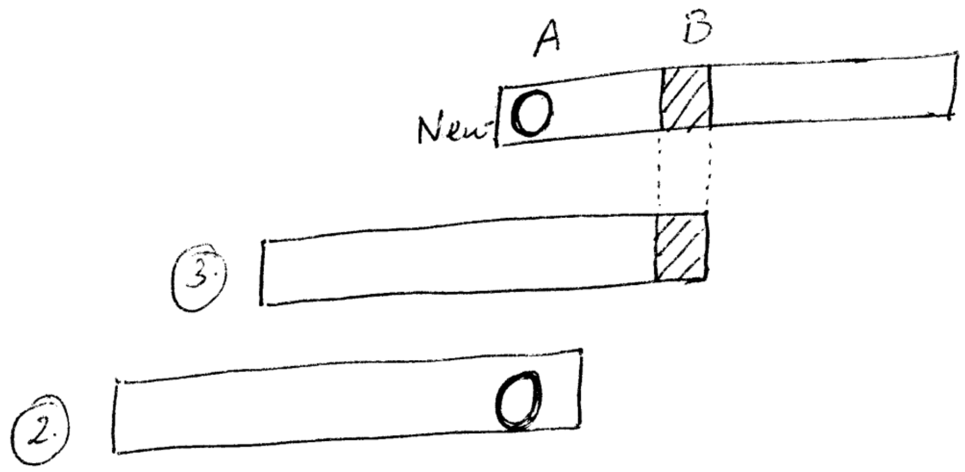

Figure 9

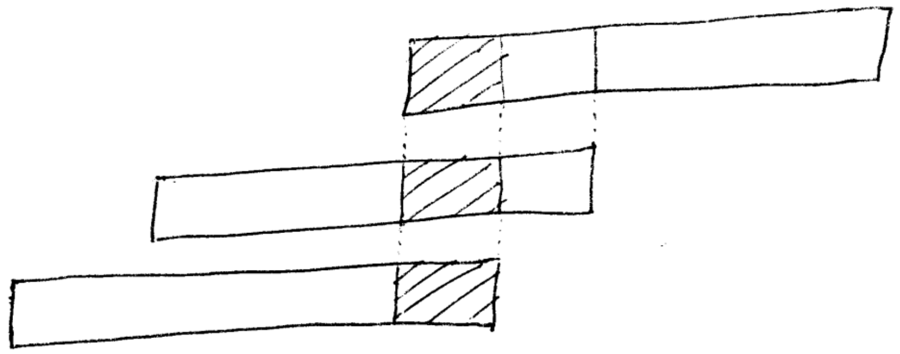

Figure 10

Resultant of elimination: we have some remaining candidates which are pair wise consistent.

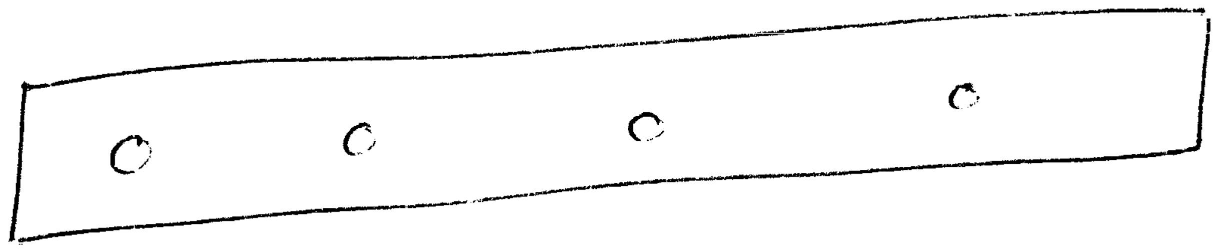

Figure 11

Verification

Consider a text with only those candidates which have passed the dueling. Number the text starting at a candidate's location and restarting every time we cross a candidate. Figure 12 gives a pictographic description of this method. The numbers circled represent candidates in the text.

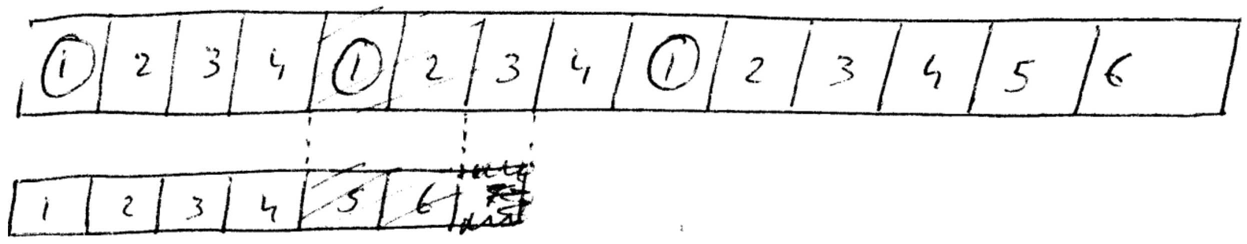

Figure 12

As it is clear from figure 12, if a pattern has to exist at both the first and second location, then p[1] = p[5] and p[2] = p[6].

Since we start counting as soon as we cross a candidate, we can never run out of numbers. Throughout the text we have to compare the element at i(location in the text) to p[i] and decide as follows:

$$\text{write} \begin{cases} y & \text{if they are equal} \\ n & \text{if not} \end{cases}$$

Figure 13

After tagging the text with 'y' and 'n', we run what is called a "wave" to eliminate the bad candidates.

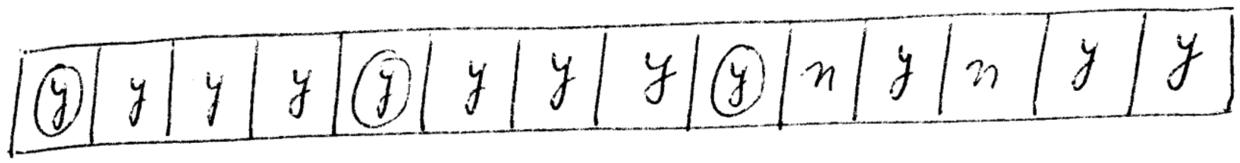

Figure 14

Wave: From the right most 'n', start numbering backwards. Start counting again if you cross an 'n'. All candidates who run into a wave would have to die. Check the remaining candidates for an exact match. In figure 14, the 1st candidate is the only candidate who survives the wave.

2 Dimensional Matching

The dueling technique we discussed for 1D can be applied efficiently to 2D as well. The naïve method however would be to match every character from text to every character in the pattern.

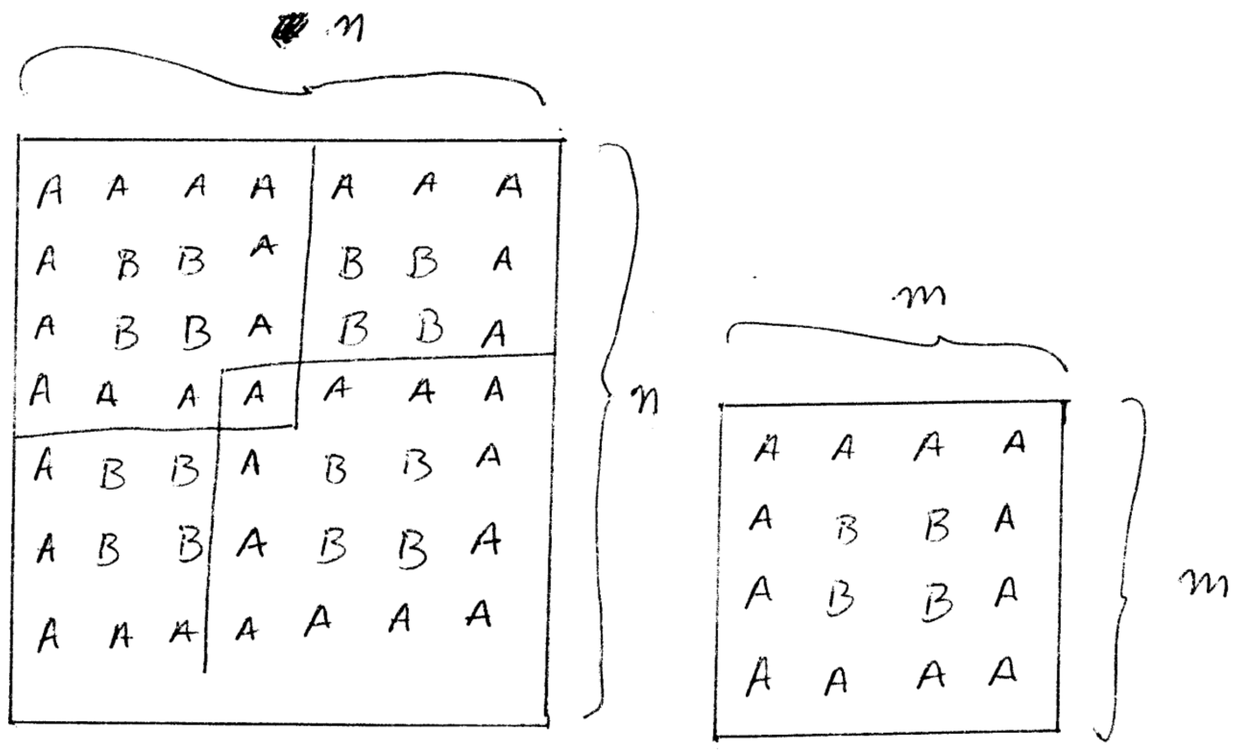

Fig 1

The time for such a comparison is O(n^2m^2), which is equal to O(nm) in 1D.

Another algorithm that one can think of to perform this task is Bird-Baker method (76,77). Refer to the lecture 2 scribe for a detailed description of this algorithm.

Coming back to dueling, it appears that pattern can be found in a text in linear time in 2D as well. So, to start with, how do we build a witness table for a pattern in 2 dimensions? As we know from 1 dimension, witness table analyses an overlap of patterns in a particular direction.

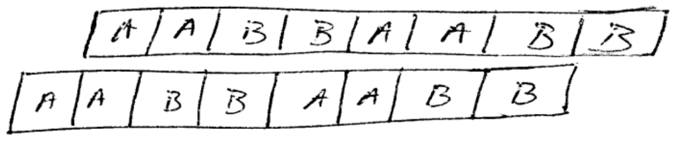

Fig 2

Figure 3 shows four directions in which a pattern can overlap itself.

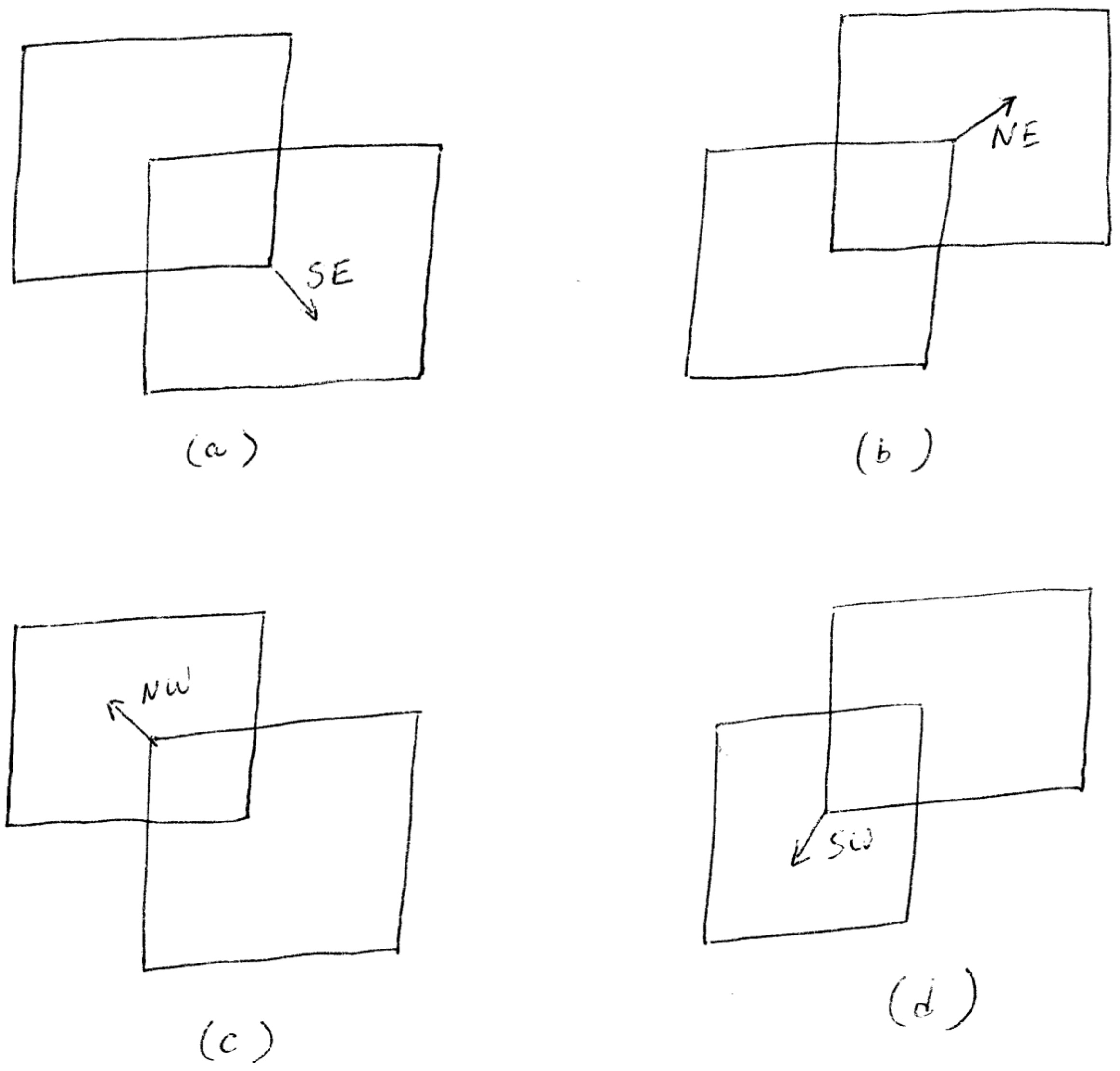

Fig 3

Does this mean we need four witness tables? No, a closer look at figure 3 shows that 3(a), 3(c) and 3(b), 3(d) are in fact the same thing. So, essentially, we need only two witness tables.

Constructing a witness table in 2D

For each position (i, j), fill in one of the positions within the overlap where there is a conflict. If there is no conflict, put a $*$ in that location.

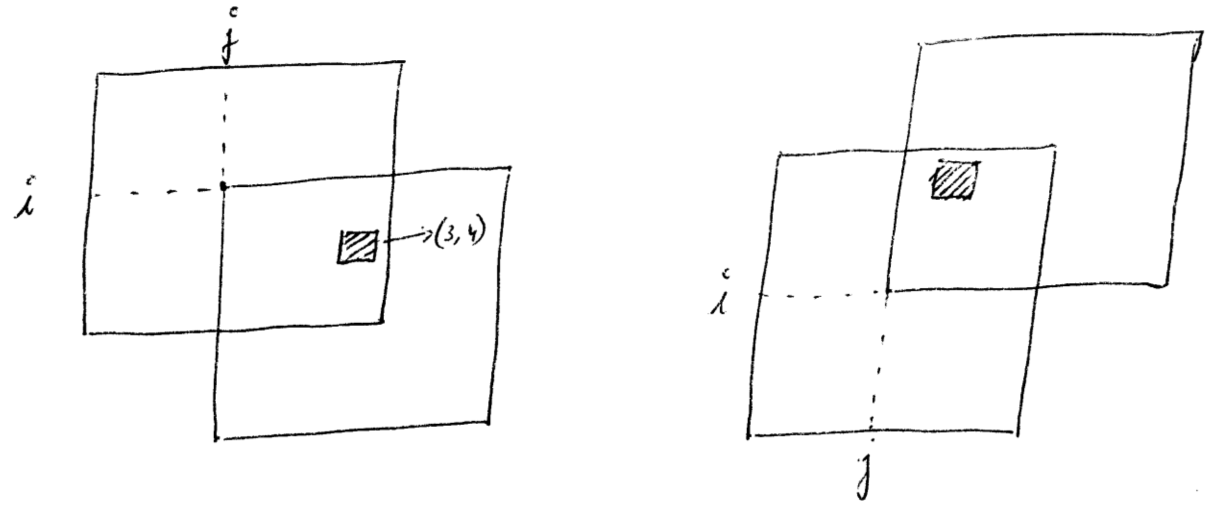

Fig 4

Text Scanning

Like in 1D, text scanning in 2D consists of two steps:

1. Eliminating conflicting sources
2. Verifying remaining candidates

Eliminating the conflicting sources:

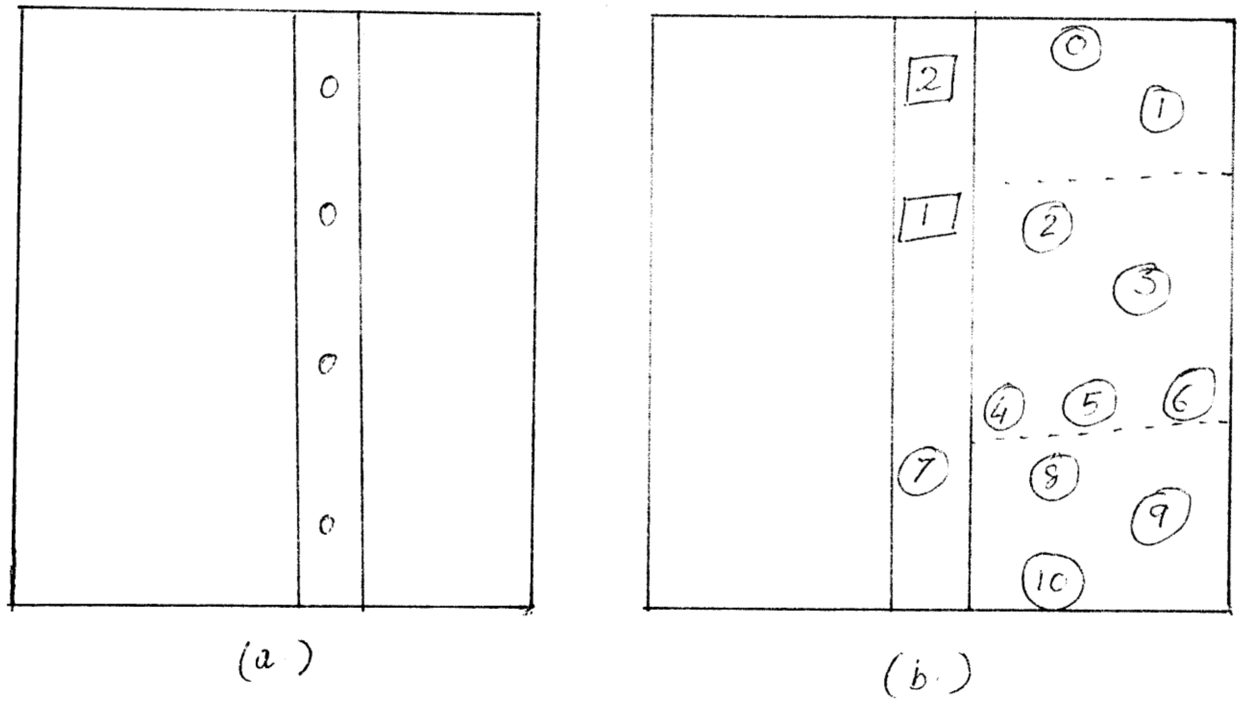

Fig 5

To do this, we start by dueling between candidates within a column starting at the n-m+1 column as shown in figure 5(a). Since this is the first column we are checking, we need not worry about previous candidates at this stage. Figure 5(b) is a general case, where we are on column I, and have 3 new candidates (7, 2, 1) in that column. We need to check for agreement between these 3 and the previous candidates to decide whether to eliminate them or not.

Let us assume $\boxed{7}$ agree with $\boxed{8}$, $\boxed{9}$ and $\boxed{10}$. This means a pattern starting at $\boxed{7}$ agrees in its overlap with $\boxed{8}$, $\boxed{9}$ and $\boxed{10}$. We claim that in this case, $\boxed{1}$ does not have to duel with $\boxed{8}$, $\boxed{9}$ and $\boxed{10}$. This is so because of the transitivity of equality.

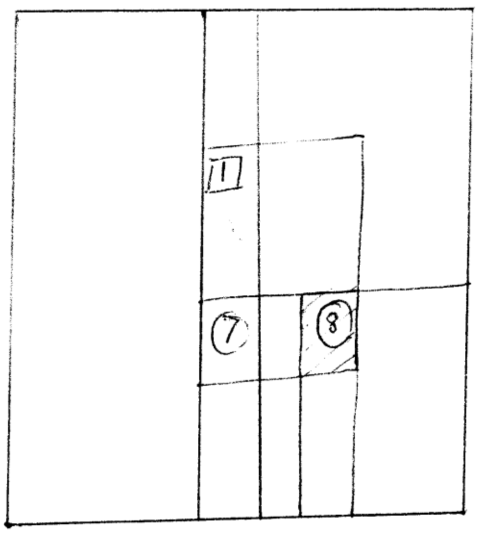

Fig 6

Figure 6 shows that overlap between with $\boxed{1}$ and $\boxed{8}$ is a subset of overlap between $\boxed{7}$ and $\boxed{8}$ substantiating our claim above.

Also, $\boxed{1}$ won't be checked with $\boxed{0}$ and $\boxed{1}$ because they are beyond its scope. The case which is left to be investigated is the duel between $\boxed{1}$ and $\boxed{2}$, $\boxed{3}$, $\boxed{4}$, $\boxed{5}$ and $\boxed{6}$. We shall start with the left most candidate to the right of $\boxed{1}$, which is $\boxed{4}$ in this case. The following cases arise in the duel between $\boxed{1}$ and $\boxed{4}$:

1. $\boxed{1}$ dies: $\boxed{1}$ pays for the duel and then we move to the next candidate ($\boxed{2}$ in this case). Confines to linearity.
2. $\boxed{4}$ dies: The candidate $\boxed{4}$ pays for the duel between $\boxed{1}$ and $\boxed{5}$, which does not hurt linearity either.
3. $\boxed{1}$ agrees with $\boxed{4}$: If $\boxed{1}$ and $\boxed{4}$ agree, we need not duel between $\boxed{1}$, $\boxed{5}$ and $\boxed{6}$ because of the transitivity of equality which is further explained below.

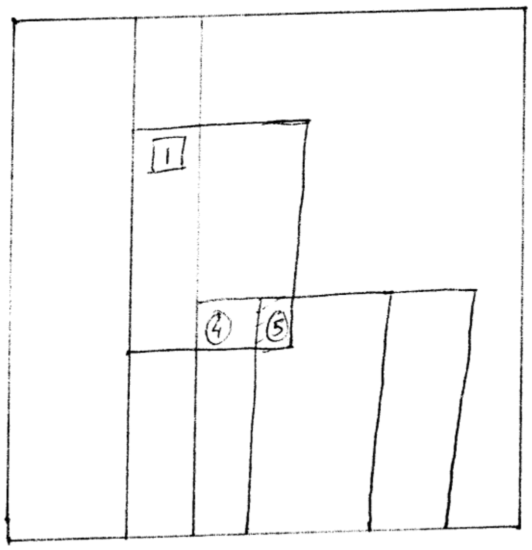

Fig 7

Therefore, if $\boxed{1}$ and $\boxed{4}$ agree, we can move ahead to the row above. So, since if the two candidates agree, we move to the row above it, this can happen in O(n) time in a column and for the entire matrix, it would be O(n^2)

Therefore, in all cases, the time is linear O(n^2).

After we finish filtering candidates in SE direction, we need to use the second witness table to verify candidates in NE direction as well. However, we only need to conduct duel between the remaining candidates.

Verification

Figure 8 shows how the scenario would be after eliminating false candidates.

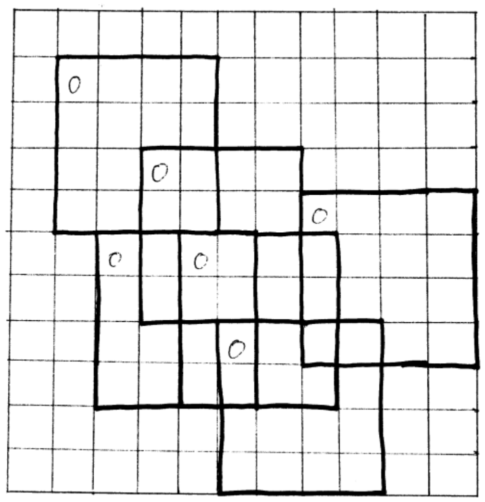

Fig 8

In each of these matrices we start numbering each cells of the first row (of each matrix) starting from the top left corner. Whenever we come across another candidate, we restart the counting. This looks like what is shown in figure 9.

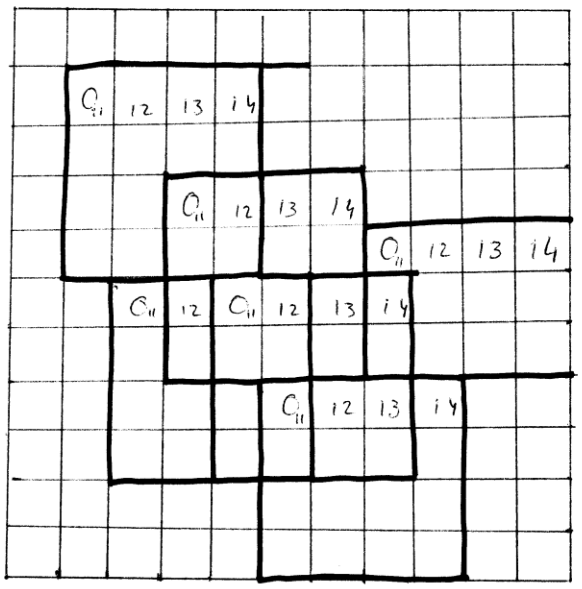

Fig 9

Now, in each column, start numbering downward following the first row. Again, when we cross a candidate, we restart our numbering. After this, our matrices would look something like in figure 10.

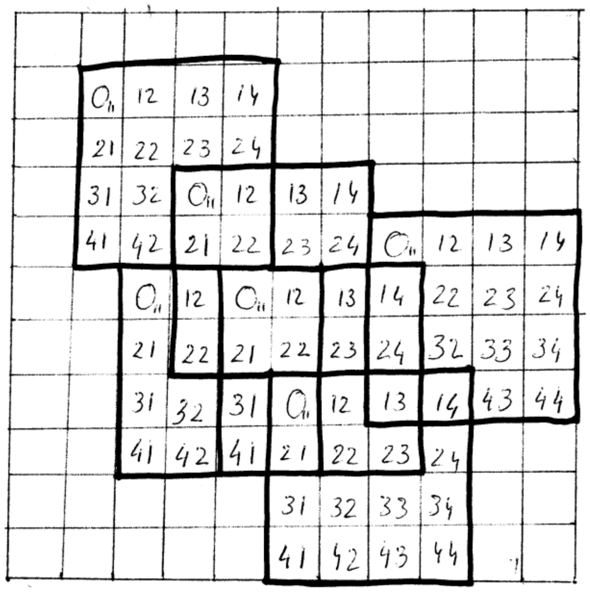

Fig 10

From here on we will follow the same technique we used in 1D. For all the numbered locationsin the matrices, we compare the content with P[i, j]: i,j being the row and column number in the text. If they are equal, we replace the number with a 'y' and if not we replace the number by 'n'.

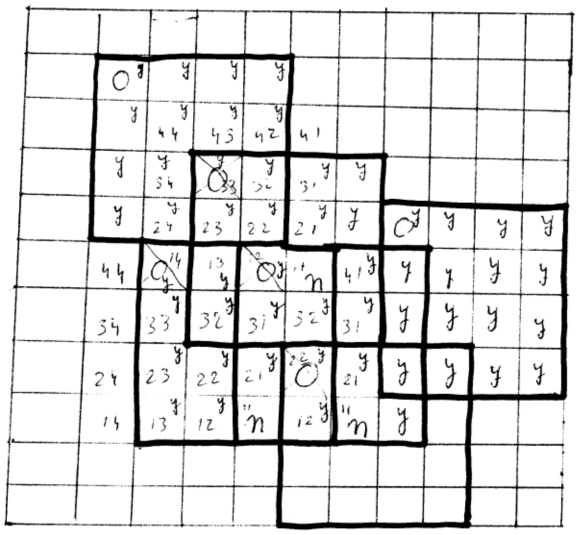

Fig 11

Starting at the rightmost n, we begin numbering row wise towards left hand side. After a row is complete, we then fill the columns in the same way as we did while numbering initially. Only this time, we start numbering when we cross an ‘n’. The idea is, is there is a mismatch at some position, a candidate which spans over this location cannot be a true candidate.

So finally, those candidates that do not ‘run under’ the numbering are qualified for final verification with the original pattern.