W06.5: Fingerprinting

CS3230 AY21/22 Sem 2

Changelog (after recording)

- Fixed typo in false positive analysis of Q4 → In the video T was used, but T is the entire text. The correct one should P, where P is the pattern.

Last Updated on: 21 Feb 2022, 12:15AM

Administrative Reminders

- Tutorial in Week 7 depends on tutors -- no attendance taken
- Tutorial in Week 8 cancelled

Table of Contents

Click on the link to jump to the relevant sections!

Question 1: Communication Complexity

Question 1

Alice and Bob are serving SHN in two separate rooms. They can only communicate through SMS, and they **get charged $1 for every bit** they transmit to each other.

Question 1

Alice and Bob are serving SHN in two separate rooms. They can only communicate through SMS, and they **get charged \$1 for every bit** they transmit to each other.

Alice has an integer x and Bob has integer y, **both nonnegative and less than 2^n**. Bob wants to know whether $x = y$.

Question 1

Alice and Bob are serving SHN in two separate rooms. They can only communicate through SMS, and they **get charged $1 for every bit** they transmit to each other.

Alice has an integer *x* and Bob has integer *y*, **both nonnegative and less than 2^n**. Bob wants to know whether $x = y$.

Alice sends *x* to Bob, and Bob compares *x* to *y*.

What is the worst-case cost for the communication between Alice and Bob?

Question 1

Alice has 1801, and Bob has 999

$x = 1801$

$y = 999$

Question 1

Alice sends x to Bob

$x = 1801$

$x = 1801$

$y = 999$

Question 1

Cost of sending this number?

$x = 1801$

$x = 1801$

$y = 999$

Question 1 (answer)

Key Fact: $0 \le x < 2^n$

Question 1 (answer)

Key Fact: $0 \leq x < 2^n$

Binary to Decimals

From prerequisite revision

If you have *n* bits, you can represent integers in decimals in the range of $[0..2^n-1]$.

(Example) With 4 bits:

0000: 0	1000: 8
0001: 1	1001: 9
0010: 2	1010: 10
0011: 3	1011: 11
0100: 4	1100: 12
0101: 5	1101: 13
0110: 6	1110: 14
0111: 7	1111: 15

$2^4 - 1$

Question 1 (answer)

Key Fact: $0 \leq x < 2^n$

Therefore cost: $\Theta(n)$

Binary to Decimals

From prerequisite revision

If you have n bits, you can represent integers in decimals in the range of $[0..2^n-1]$.

(Example) With 4 bits:

0000: 0	1000: 8
0001: 1	1001: 9
0010: 2	1010: 10
0011: 3	1011: 11
0100: 4	1100: 12
0101: 5	1101: 13
0110: 6	1110: 14
0111: 7	1111: 15

$2^4 - 1$

(With n-1 bits, you can only represent 2^{n-1} integers, not enough for the range. On the other hand, n bits can represent all the 2^n integers)

Question 2: (Better) Communication Complexity

Question 2

Alice and Bob are serving SHN in two separate rooms. They can only communicate through SMS, and they **get charged $1 for every bit** they transmit to each other.

Alice has an integer x and Bob has integer y, **both nonnegative and less than 2^n**. Bob wants to know whether $x = y$.

Alice randomly chooses a prime number p, computes $a = x \bmod p$, sends p and a to Bob. Bob computes $b = y \bmod p$. Bob then compares a and b.

What is the cost if they want the false positive probability to be < 1%?

Question 2

Alice has 1801, and Bob has 999

Suppose random p = 131

$x = 1801, p = 131$

$y = 999$

Question 2

`a = 1801 mod 131 = 98`

$x = 1801, p = 131$

$a = 98$

$y = 999$

Question 2

`Alice sends a = 98 AND p = 131`

$a = 98, p = 131$

$x = 1801, p = 131$

$a = 98$

$y = 999$

Question 2

Bob computes $b = 999 \bmod 131 = 82$

$a = 98, p = 131$

$x = 1801, p = 131$

$a = 98$

$y = 999$

$b = 82$

Question 2

Bob: $a \neq b \rightarrow x \neq y$

$a = 98, p = 131$

$x = 1801, p = 131$

$a = 98$

$y = 999$

$b = 82$

Question 2

What is the cost of sending the number and prime?

x ≠ y!

$a = 98, p = 131$

$x = 1801, p = 131$

$a = 98$

$y = 999$

$b = 82$

Q2 (Ans)

Analyse using the division hash from the lecture! Suppose Alice chooses *p* from a range *{1, … , K}*.

For example, *K = 200*. Alice chooses a prime from *{1, … , 200}* and get *p = 131*

Q2 (Ans)

Analyse using the division hash from the lecture! Suppose Alice chooses *p* from a range *{1, … , K}*.

$$\Pr[a = b]$$

Probability of false positive:
x ≠ y, but their hashes *a = b*

Q2 (Ans)

Analyse using the division hash from the lecture! Suppose Alice chooses *p* from a range *{1, … , K}*.

Let $z = x - y$:

$$\Pr[a = b] = \Pr[z = 0 \ (mod\ p)]$$

The prime p is one of the prime factors of (x - y)

Q2 (Ans)

Claim: If $0 \le x < y < 2^b$, then:

$$\Pr_p[h_p(x) = h_p(y)] < \frac{b \ln K}{K}.$$

Analyse using the division hash from the lecture! Suppose Alice chooses *p* from a range *{1, ... , K}*.

Let $z = x - y$:

n because *x* and *y* < 2^n

$$\Pr[a = b] = \Pr[z = 0\ (mod\ p)] < \frac{n \lg K}{K}$$

From the lecture!

Q2 (Ans)

Claim: If $0 \le x < y < 2^b$, then:

$$\Pr_p[h_p(x) = h_p(y)] < \frac{b \ln K}{K}.$$

Analyse using the division hash from the lecture! Suppose Alice chooses *p* from a range *{1, ... , K}*.

Let *z = x - y*:

$$\Pr[a = b] = \Pr[z = 0\ (mod\ p)] < \frac{n \lg K}{K}$$

Goal: False positive rate should be less than 1%

$$\frac{n\lg(K)}{K}$$

Question 2

Goal: nlg(K) / K < 1%

What value of K will give you < 1/100?

Question 2

Goal: nlg(K) / K < 1%

$$\frac{n \lg(K)}{K}$$

What value of K will give you < 1/100?

Similar intuition. See forum

Re: Lecture 6: Intuition behind the proof for Equality Check Analysis

Posted by **Warut Suksompong** on 18 Feb 2022 8:29 pm.

Thanks for asking this! Yes, that's correct -- we want the term $1/(100n)$ on the right, in order to apply the union bound in the next line and get $1/100 = 1\%$.

So we ask ourselves: What value of K would make $m \cdot \frac{\ln K}{K} < \frac{1}{100n}$? Note that if the term $\ln K$ was not there, we could just choose $K = 200mn$ and get $\frac{m}{K} = \frac{1}{200n} < \frac{1}{100n}$. But since the term $\ln K$ is there, $K = 200mn$ is no longer sufficient, because there will be a term $\ln(200mn)$ in the numerator. That's why we add a factor $\ln(200mn)$ to K to help cancel this term, and then do the math to show that this is sufficient.

Hope this helps!

Choose $K = 200n \lg(200n)$

$$\frac{n \lg(K)}{K}$$

Question 2

Goal: nlg(K) / K < 1%

Choose $K = 200n \lg(200n)$

$$\frac{n \lg(K)}{K} = \frac{n \lg(200n \lg(200n))}{200n \lg(200n)}$$

Question 2

Substitute K in

Choose $K = 200n \lg(200n)$

$$\frac{n\lg(K)}{K} = \frac{\cancel{n}\lg(200n\lg(200n))}{200\cancel{n}\lg(200n)}$$

$$= \frac{1}{200} \cdot \frac{\lg(200n\lg(200n))}{\lg(200n)}$$

Question 2

```
Cancel n, and factorise 1/200
```

Choose $K = 200n \lg(200n)$

$$\frac{n \lg(K)}{K} = \frac{n \lg(200n \lg(200n))}{200n \lg(200n)}$$

$$= \frac{1}{200} \cdot \frac{\lg(\boxed{200n}\,\boxed{\lg(200n)})}{\lg(200n)}$$

$$= \frac{1}{200} \cdot \frac{\lg(\boxed{200n}) + \lg\boxed{\lg(200n)}}{\lg(200n)}$$

Question 2

lg(ab) = lg(a) + lg(b)

Choose $K = 200n \lg(200n)$

$$
\begin{aligned}
\frac{n \lg(K)}{K} &= \frac{n \lg(200n \lg(200n))}{200n \lg(200n)} \\
&= \frac{1}{200} \cdot \frac{\lg(200n \lg(200n))}{\lg(200n)} \\
&= \frac{1}{200} \cdot \frac{\boxed{\lg(200n)} + \lg\lg(200n)}{\boxed{\lg(200n)}} \\
&= \frac{1}{200} \cdot \left(1 + \frac{\lg\lg(200n)}{\lg(200n)}\right)
\end{aligned}
$$

Question 2

Cancel the lg(200n)

Choose $K = 200n \lg(200n)$

$$
\begin{aligned}
\frac{n \lg(K)}{K} &= \frac{n \lg(200n \lg(200n))}{200n \lg(200n)} \\
&= \frac{1}{200} \cdot \frac{\lg(200n \lg(200n))}{\lg(200n)} \\
&= \frac{1}{200} \cdot \frac{\lg(200n) + \lg \lg(200n)}{\lg(200n)} \\
&= \frac{1}{200} \cdot \left(1 + \boxed{\frac{\lg \lg(200n)}{\lg(200n)}}\right) \\
&\boxed{<} \frac{1}{200} \cdot 2
\end{aligned}
$$

< 1

Question 2

The ratio of lglg and lg is < 1

Intuition: Doing lg one more time will make the number even smaller

Choose $K = 200n \lg(200n)$

$$\begin{aligned}
\frac{n \lg(K)}{K} &= \frac{n \lg(200n \lg(200n))}{200n \lg(200n)} \\
&= \frac{1}{200} \cdot \frac{\lg(200n \lg(200n))}{\lg(200n)} \\
&= \frac{1}{200} \cdot \frac{\lg(200n) + \lg \lg(200n)}{\lg(200n)} \\
&= \frac{1}{200} \cdot \left(1 + \frac{\lg \lg(200n)}{\lg(200n)}\right) \\
&< \frac{1}{200} \cdot 2 \\
&= \frac{1}{100}
\end{aligned}$$

Question 2

We get nlg(K)/K < 1% as required!

Q2 (ans)

```
Both these numbers are ≤ K
```

$a = 98, p = 131$

$K = 200n\lg(200n)$

Q2 (ans)

`Recall: We are charged by the number of bits`

$a = 98, p = 131$

$K = 200n\lg(200n)$

Takes lg(K) bits to represent:

$$\lg(K) = \lg(200n\lg(200n))$$

Q2 (ans)

`Recall: We are charged by the number of bits`

a = 98, p = 131

K = 200nlg(200n)

Takes lg(K) bits to represent:

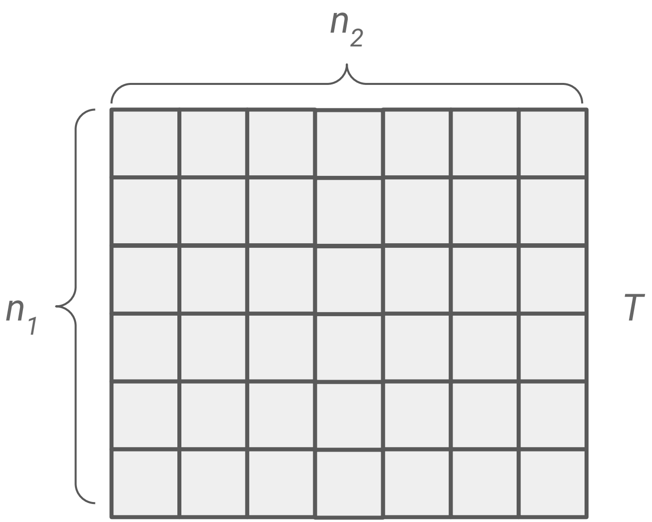
$$\begin{aligned}\lg(K) &= \lg(200n\lg(200n)) \\ &= \lg(200) + \lg(n) + \lg\lg(200n) \\ &= \Theta(\lg n)\end{aligned}$$

Communication cost

Question 3: 2D Pattern Matching (Naive)

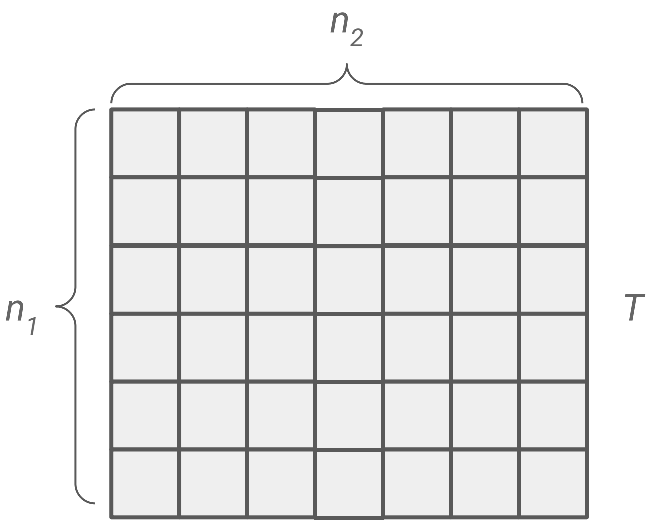
Question 3

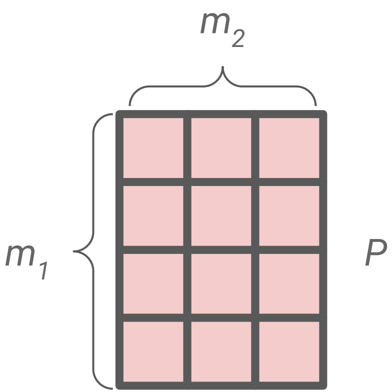
We have a text string *T* which is an n_1 x n_2 sized rectangle



Question 3

We have a text string T which is an n_1 x n_2 sized rectangle, and the pattern string P is an m_1 x m_2 sized rectangle. Here $m_1 \leq n_1$ and $m_2 \leq n_2$.

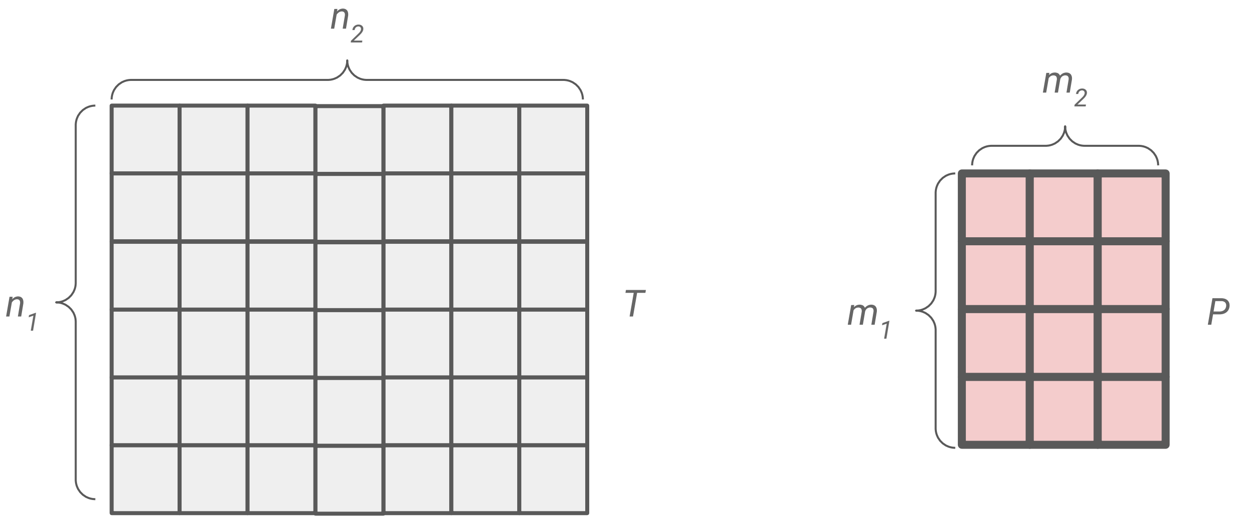




Question 3

We have a text string T which is an n_1 x n_2 sized rectangle, and the pattern string P is an m_1 x m_2 sized rectangle. Here $m_1 \le n_1$ and $m_2 \le n_2$.

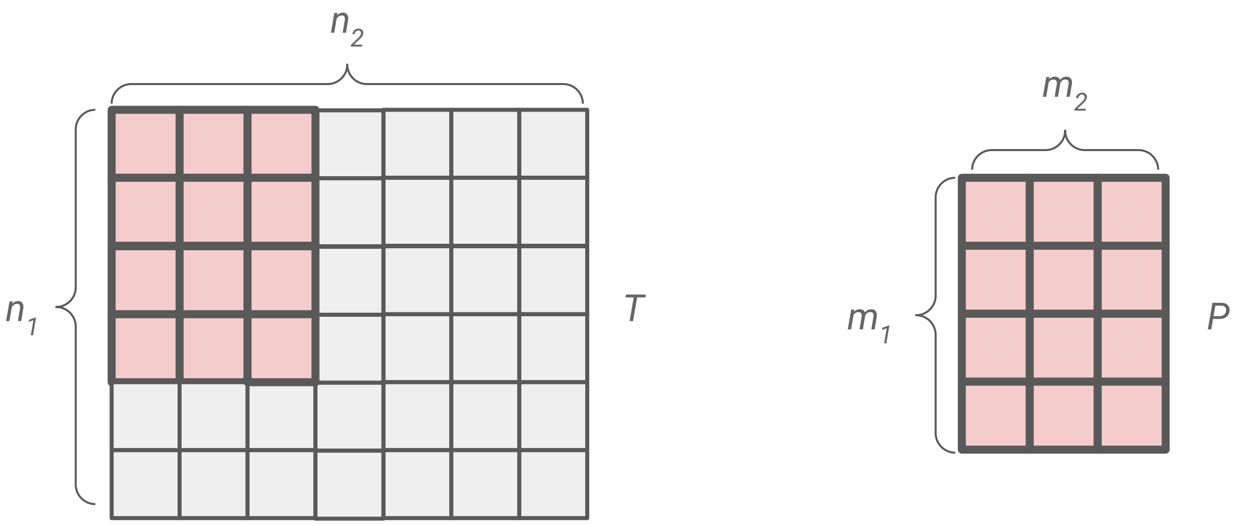
What is the time complexity for the naive algorithm that checks whether each m_1 x m_2 sized block in T equals P?



Question 3 (Ans)

Brute-forcing the pattern once: $\Theta(m_1m_2)$

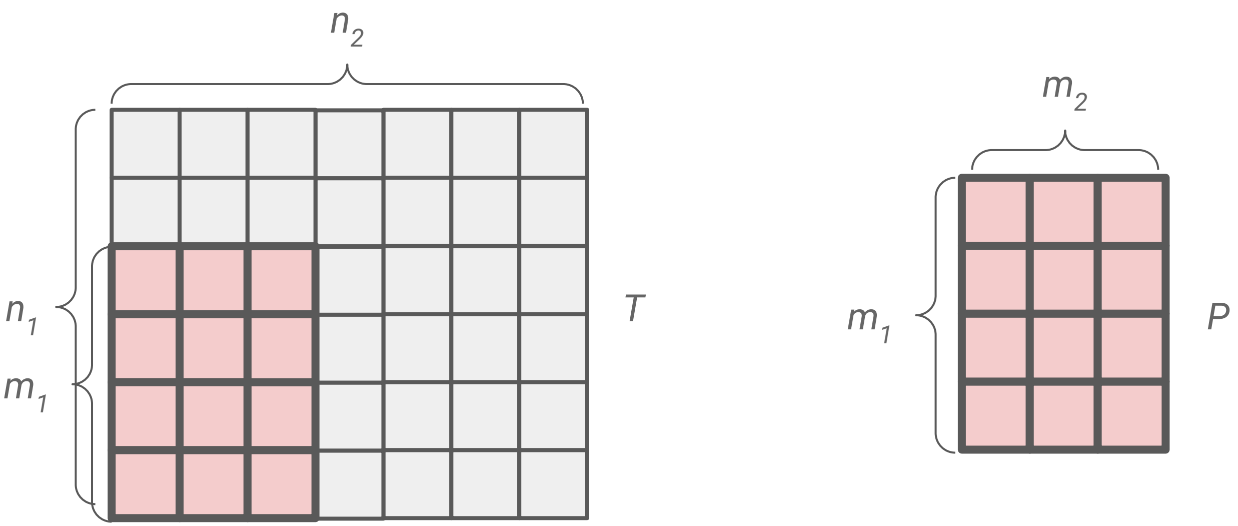
How many times do we need to brute force the pattern?



One brute force: $\theta(m_1 m_2)$

Question 3 (Ans)

Look at the top left corner! How much can you move it vertically?

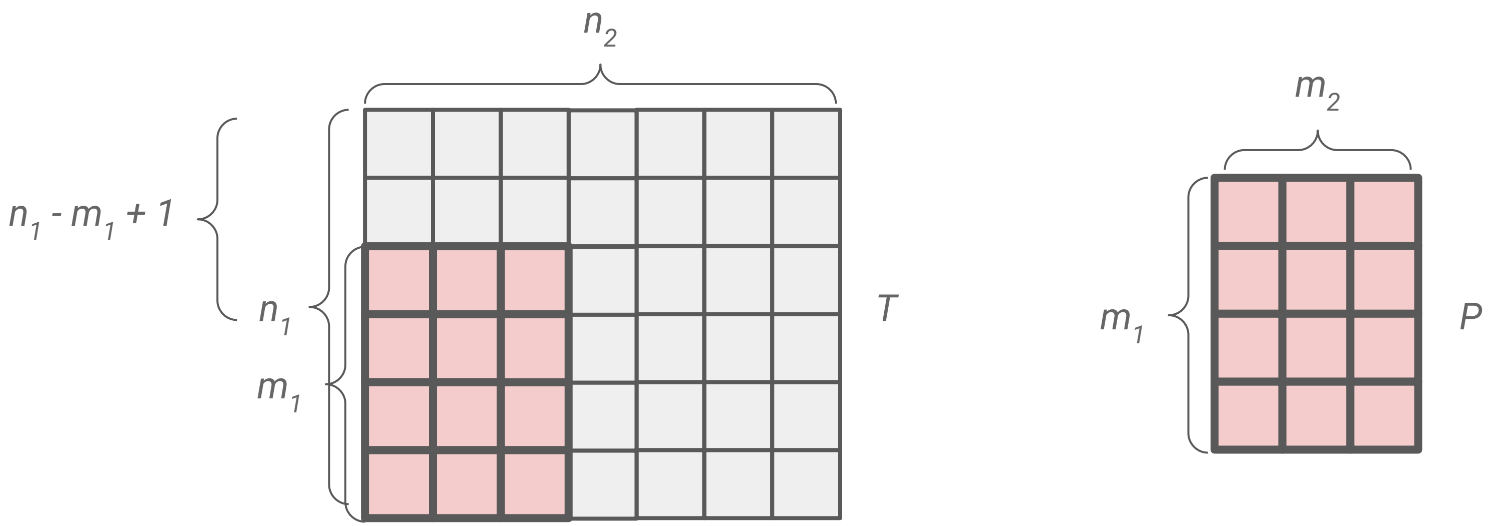


One brute force: $\theta(m_1 m_2)$

Question 3 (Ans)

Look at the top left corner! How much can you move it vertically?

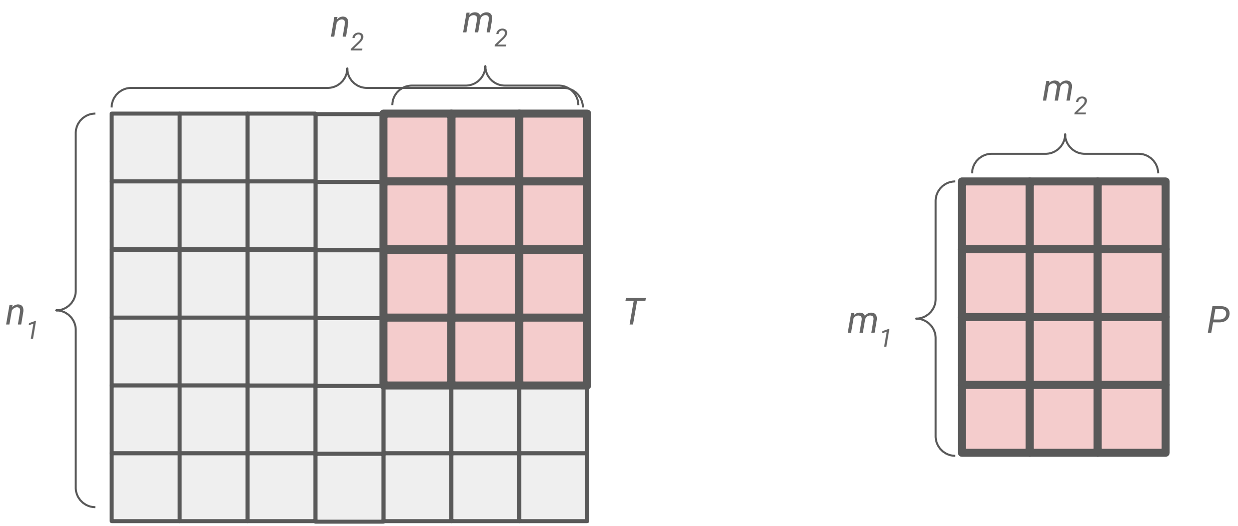
$n_1 - m_1 + 1$ vertical positions



One brute force: $\theta(m_1 m_2)$
$n_1 - m_1 + 1$ vertical positions

Question 3 (Ans)

Similarly: look at the top left corner! How much can you move it horizontally?

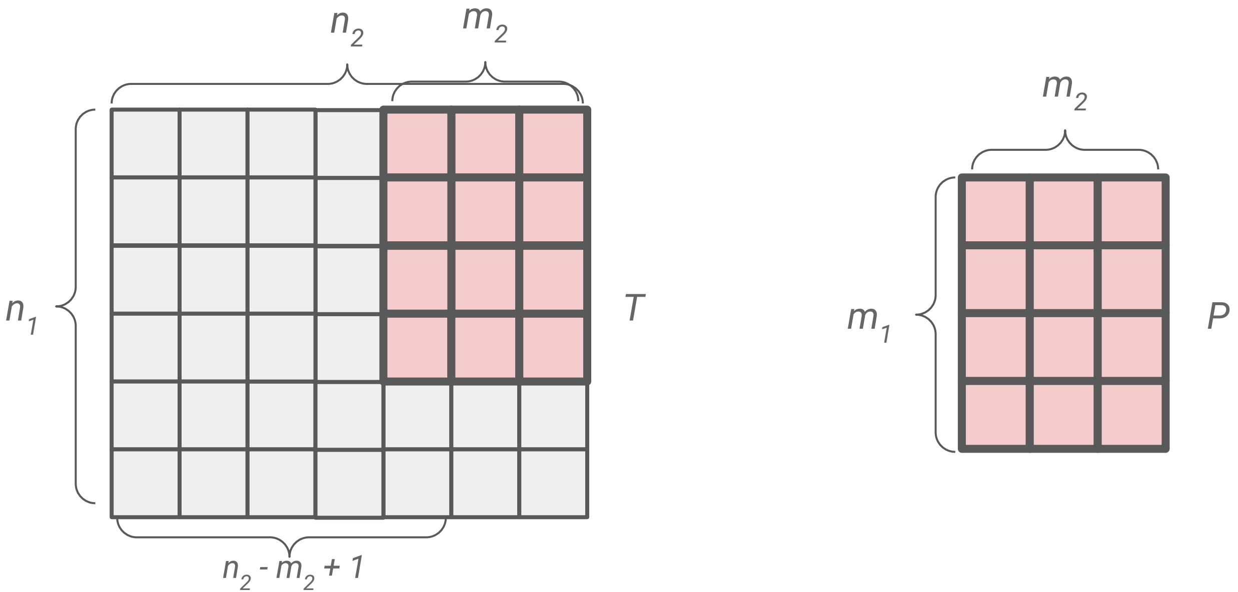


One brute force: $\theta(m_1 m_2)$
$n_1 - m_1 + 1$ vertical positions

Question 3 (Ans)

Similarly: look at the top left corner! How much can you move it horizontally?

$n_2 - m_2 + 1$ horizontal positions



One brute force: $\theta(m_1 m_2)$
$n_1 - m_1 + 1$ vertical positions
$n_2 - m_2 + 1$ horizontal positions

Question 3 (Ans)

Therefore, you need to brute force the pattern $(n_1 - m_1 + 1)(n_2 - m_2 + 1)$ times

One brute force: $\theta(m_1m_2)$
$n_1 - m_1 + 1$ vertical positions
$n_2 - m_2 + 1$ horizontal positions

Question 3 (Ans)

Therefore, you need to brute force the pattern $(n_1 - m_1 + 1)(n_2 - m_2 + 1)$ times

Since one brute force takes $\Theta(m_1m_2)$ time

Total time: $\Theta((n_1 - m_1 + 1)(n_2 - m_2 + 1)m_1m_2)$

One brute force: $\theta(m_1m_2)$
$n_1 - m_1 + 1$ vertical positions
$n_2 - m_2 + 1$ horizontal positions

Question 3 (Ans)

Therefore, you need to brute force the pattern $(n_1 - m_1 + 1)(n_2 - m_2 + 1)$ times

Since one brute force takes $\Theta(m_1m_2)$ time

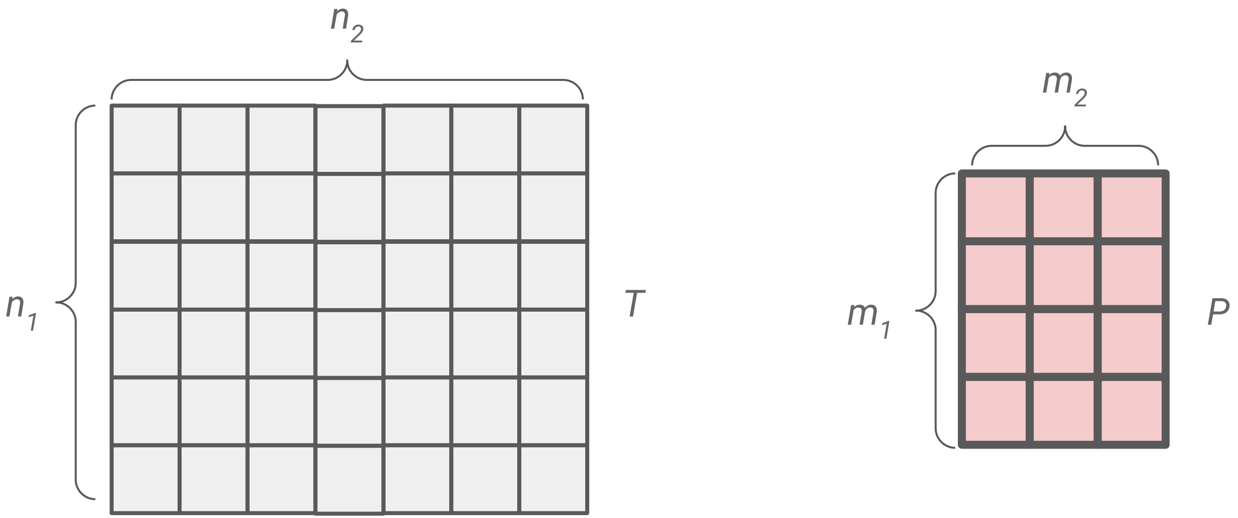
Total time: $\Theta((n_1 - m_1 + 1)(n_2 - m_2 + 1)m_1m_2) = O(n_1n_2m_1m_2)$

Question 4: 2D Pattern Matching (Karp-Rabin)

Question 4

We have a text string T which is an n_1 x n_2 sized rectangle, and the pattern string P is an m_1 x m_2 sized rectangle. Here $m_1 \leq n_1$ and $m_2 \leq n_2$.

Extend the Karp-Rabin algorithm to solve the problem in time $O(n_1 n_2)$ with 1% probability of false positive. Assume that arithmetic on integers of size $O(n_1 + n_2)$ can be done in $O(1)$ time



Q4 (Ans)

How do you hash a rectangular block of text?

Q4 (Ans)

How do you hash a rectangular block of text?

View it in a “snaking” manner:

2^5	2^4	2^3
2^2	2^1	2^0

Q4 (Ans)

How do you hash a rectangular block of text?

View it in a “snaking” manner:

2^5	2^4	2^3
2^2	2^1	2^0

1	0	1
1	1	0

- View this block as binary number 101110
- Convert to decimal and hash

Q4 (Ans)

How do you hash a rectangular block of text?

View it in a “snaking” manner:

2^5	2^4	2^3
2^2	2^1	2^0

1	0	1
1	1	0

So this block in decimal:
$2^5 + 2^3 + 2^2 + 2^1 = 46$

- View this block as binary number 101110
- Convert to decimal and hash

Recap: Rolling Hash in Karp Rabin

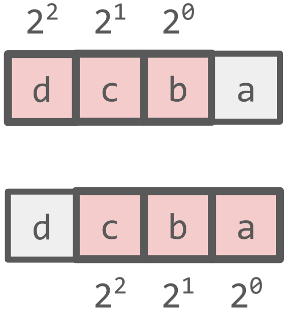
$x_1 = 2^2 \cdot d + 2^1 \cdot c + 2^0 \cdot b$

2^2	2^1	2^0	
d	c	b	a

Recap: Rolling Hash in Karp Rabin

$x_1 = 2^2 \cdot d + 2^1 \cdot c + 2^0 \cdot b$

$x_2 = 2^2 \cdot c + 2^1 \cdot b + 2^0 \cdot a$

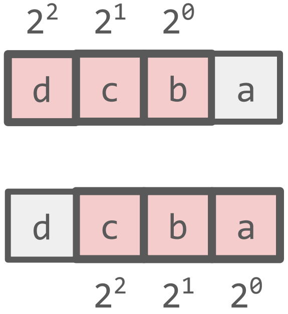


Recap: Rolling Hash in Karp Rabin

$x_1 = 2^2 \cdot d + 2^1 \cdot c + 2^0 \cdot b$

x 2

$x_2 = 2^2 \cdot c + 2^1 \cdot b + 2^0 \cdot a$



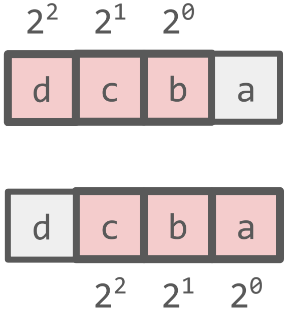
Deriving x_2 from x_1:

Recap: Rolling Hash in Karp Rabin

$x_1 = 2^2 \cdot d + 2^1 \cdot c + 2^0 \cdot b$

x 2

$x_2 = 2^2 \cdot c + 2^1 \cdot b + 2^0 \cdot a$



Deriving x_2 from x_1:

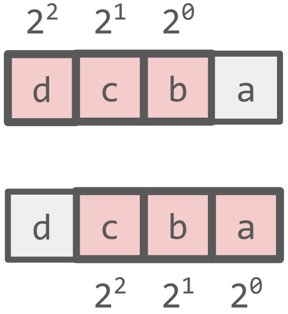
x_2
$= 2(x_1 - 2^2 \cdot d) + 2^0 a$

Recap: Rolling Hash in Karp Rabin

$$x_1 = 2^2 \cdot d + 2^1 \cdot c + 2^0 \cdot b$$

x 2

$$x_2 = 2^2 \cdot c + 2^1 \cdot b + 2^0 \cdot a$$



Deriving x_2 from x_1:

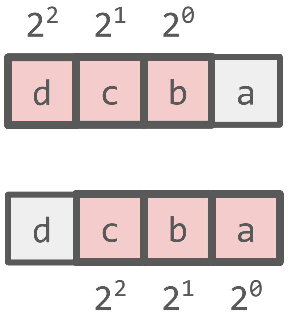
$$\begin{aligned} x_2 &\\ &= 2(x_1 - 2^2 \cdot d) + 2^0 a \\ &= 2x_1 - 2^3 \cdot d + 2^0 a \end{aligned}$$

Recap: Rolling Hash in Karp Rabin

$x_1 = 2^2 \cdot d + 2^1 \cdot c + 2^0 \cdot b$

x 2

$x_2 = 2^2 \cdot c + 2^1 \cdot b + 2^0 \cdot a$



Deriving x_2 from x_1:

x_2

$= 2(x_1 - 2^2 \cdot d) + 2^0 a$

$= 2x_1 - 2^3 \cdot d + 2^0 a$

$h(x) = x \bmod p$

Division Hashing is linear:

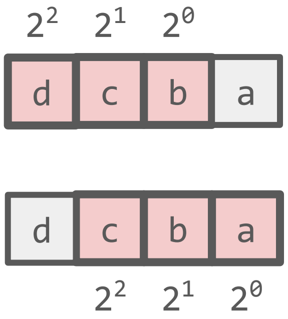
$h(x_2) = 2h(x_1) - h(2^3) \cdot d + 2^0 a \pmod p$

Recap: Rolling Hash in Karp Rabin

$$x_1 = 2^2 \cdot d + 2^1 \cdot c + 2^0 \cdot b$$

$\times\ 2$

$$x_2 = 2^2 \cdot c + 2^1 \cdot b + 2^0 \cdot a$$



Deriving x_2 from x_1:

$$\begin{aligned} x_2 &= 2(x_1 - 2^2 \cdot d) + 2^0 a \\ &= 2x_1 - 2^3 \cdot d + 2^0 a \end{aligned}$$

Now generalise this to the 2D case -- roll column by column / row by row!

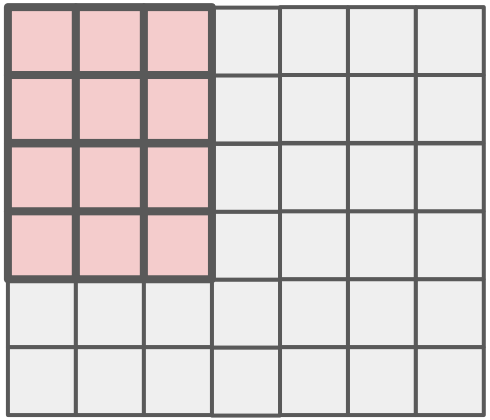
$$h(x) = x \bmod p$$

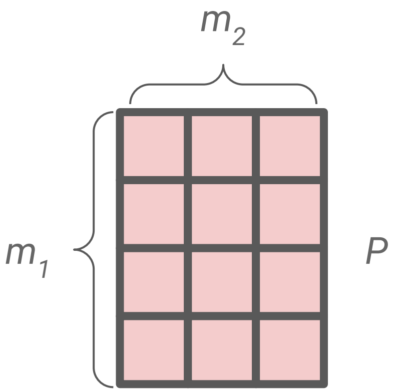
Division Hashing is linear:

$$h(x_2) = 2h(x_1) - h(2^3) \cdot d + 2^0 a \pmod p$$

Q4 High-level

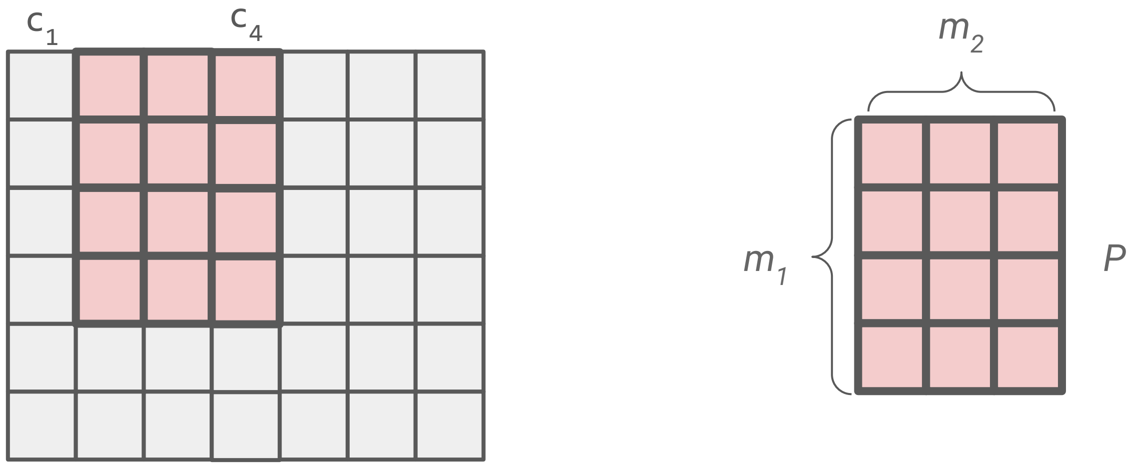
Start from top-left corner





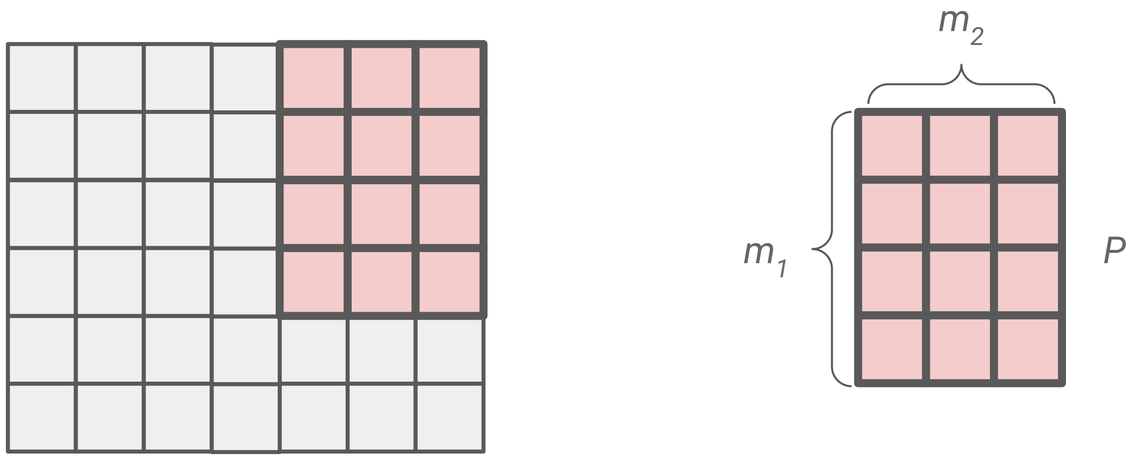
Q4 High-level

Roll by adding column 4 and removing column 1



Q4 High-level

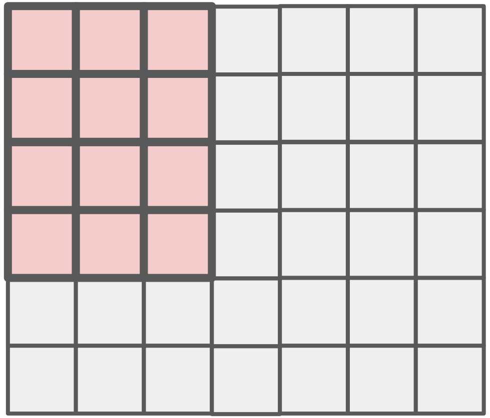
Keep rolling column by column

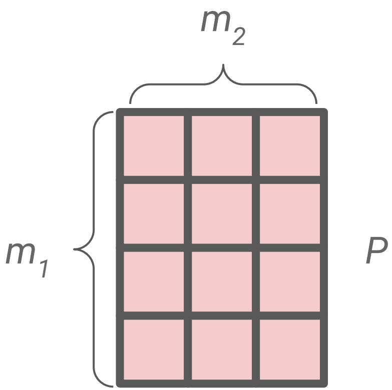


Q4 High-level

Restart, about to roll to the next row…

Restarting…

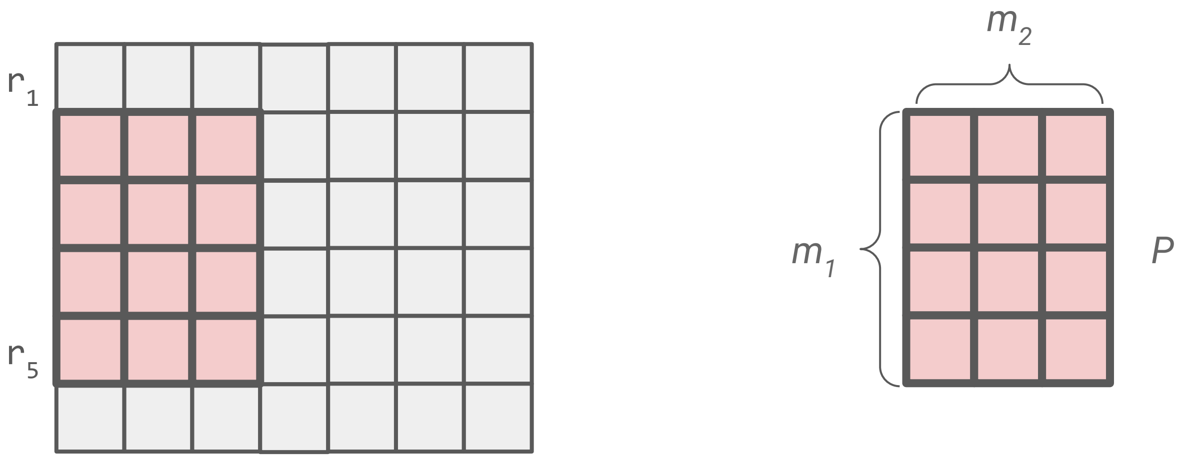




Q4 High-level

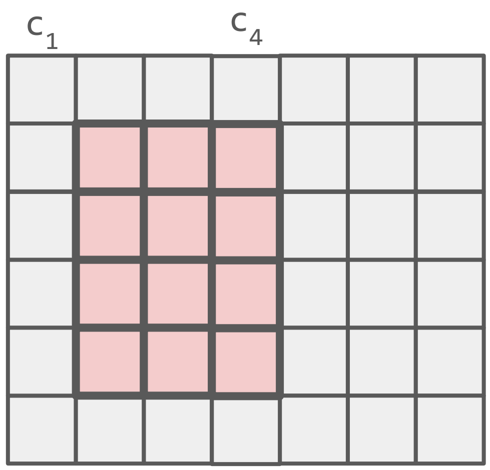
Start again from second row, by removing row 1 and adding row 5

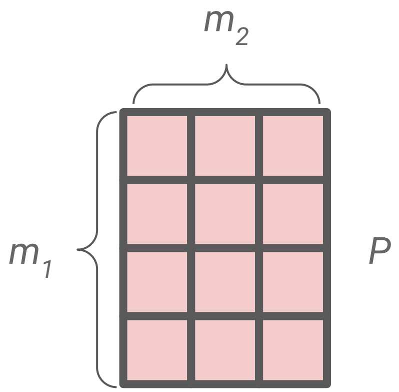
Restarting…



Q4 High-level

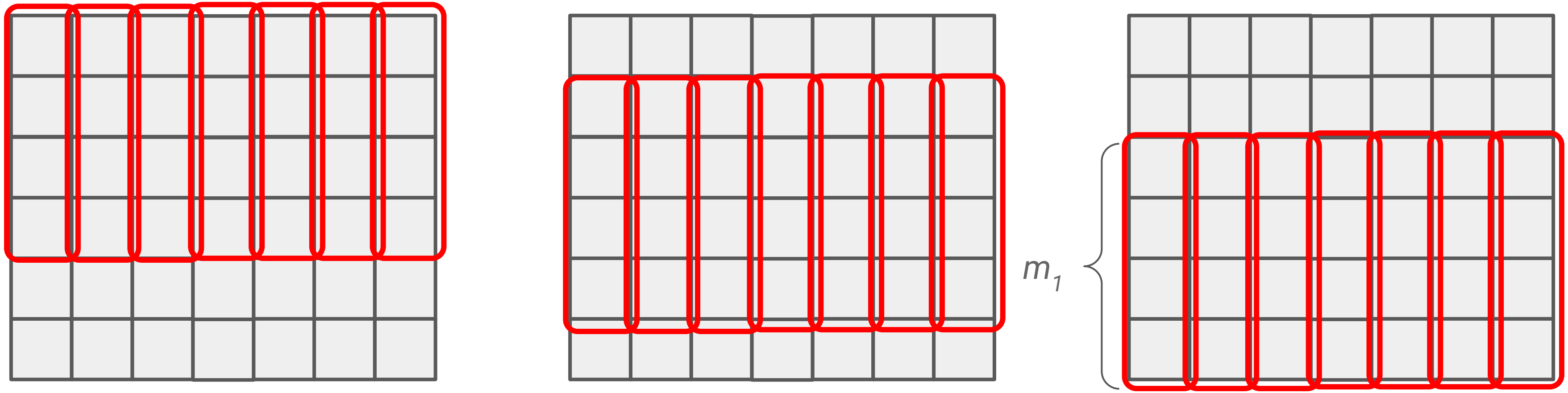
Then roll column by column again!





Which hashes do we need for 2D rolling hash?

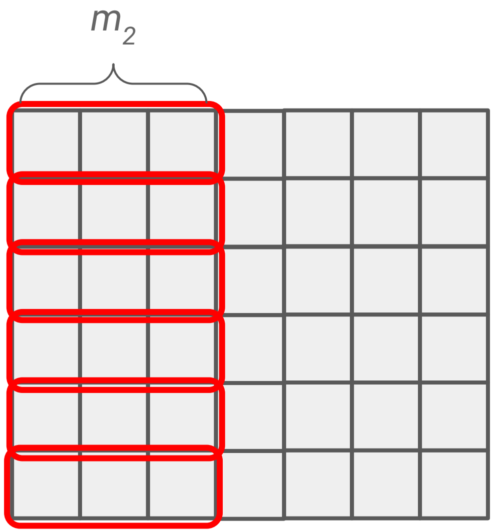
(1) To cover all horizontal movements: all the length m_1 vertical column hashes



Which hashes do we need for 2D rolling hash?

(1) To cover all horizontal movements: all the length m_1 vertical column hashes

(2) To cover vertical movement: just the first set of length m_2 horizontal row hashes

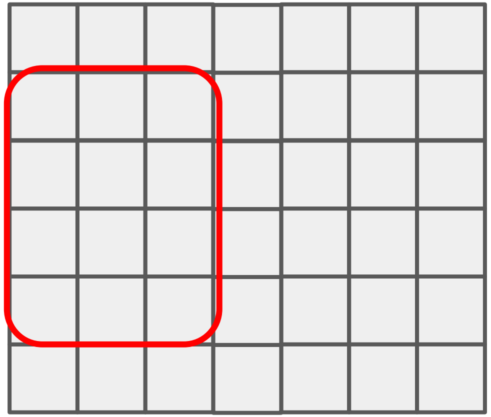


Which hashes do we need for 2D rolling hash?

(1) To cover all horizontal movements: all the length m_1 vertical column hashes

(2) To cover vertical movement: just the first set of length m_2 horizontal row hashes

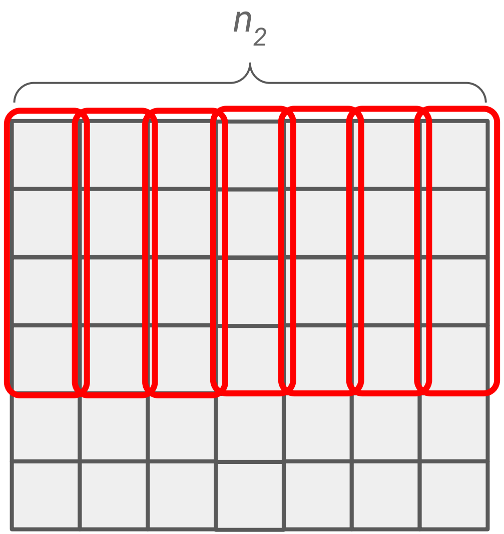
(3) The m_1 x m_2 hash for the row currently worked on -- to "restart" from left and right easily



Need to precompute all?

(1) To cover all horizontal movements: **all** the length m_1 vertical column hashes

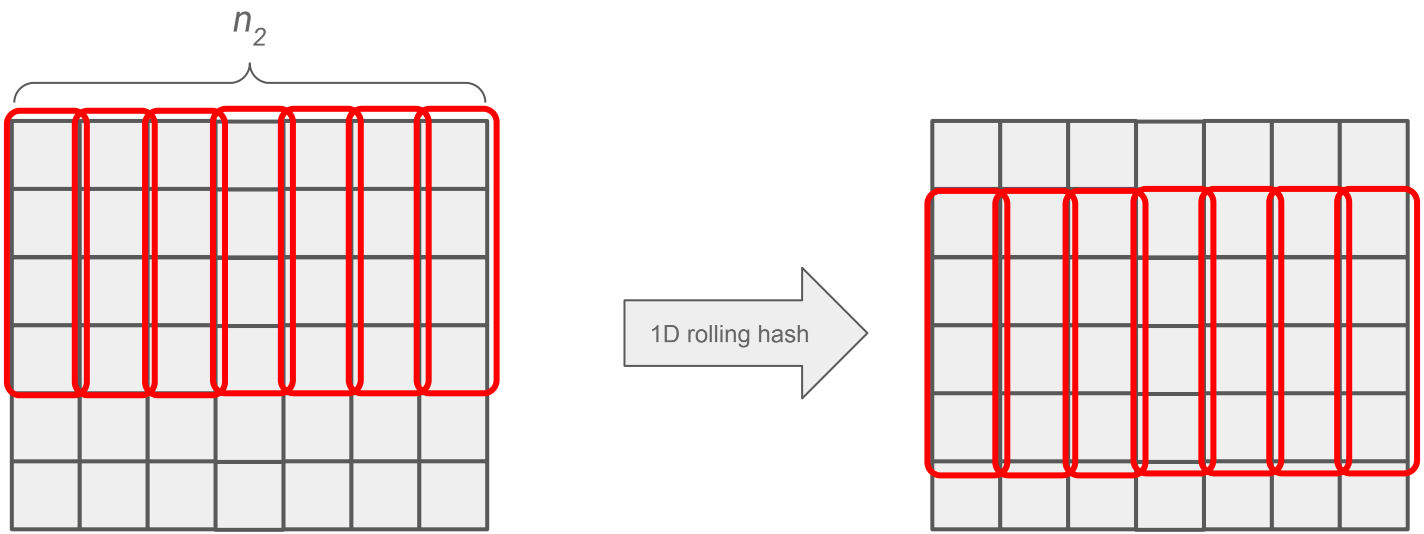
- Maintain n_2 of such hashes at a time when going column-by-column



Need to precompute all?

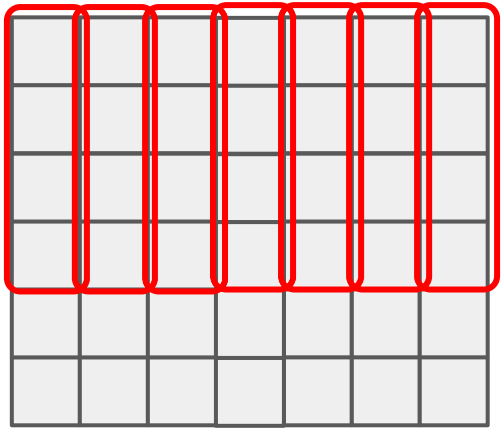
(1) To cover all horizontal movements: **all** the length m_1 vertical column hashes

- Maintain n_2 of such hashes at a time when going column-by-column
- When doing the next row, apply 1D-rolling hash to all n_2 of them



Fleshing it out

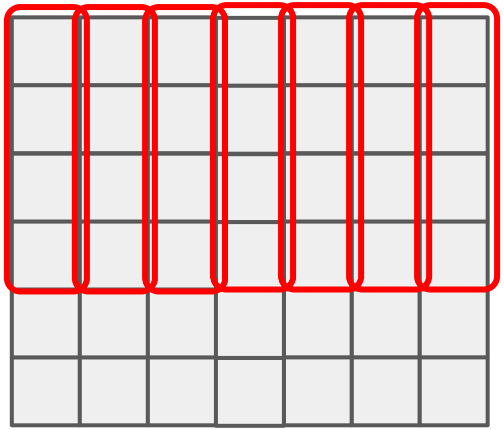
How should we hash the vertical columns?

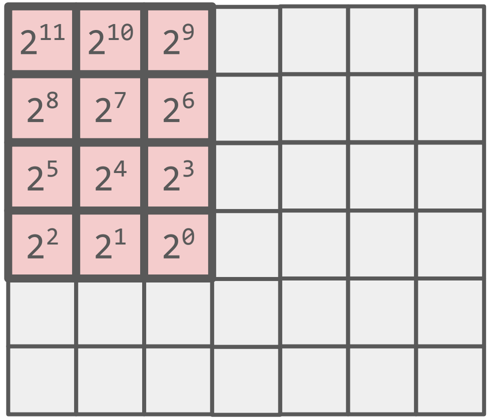


Fleshing it out

How should we hash the vertical columns?

Look at the arrangement of the **full pattern**

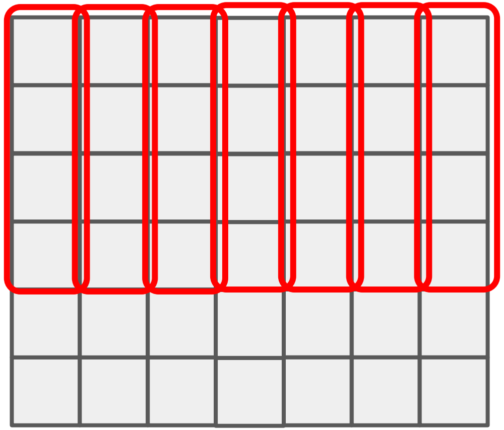


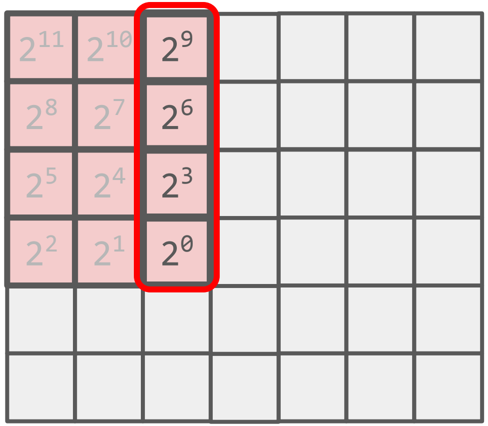


Fleshing it out

How should we hash the vertical columns?

Look at the arrangement of the **full pattern** -- take the rightmost column. It is the "simplest"

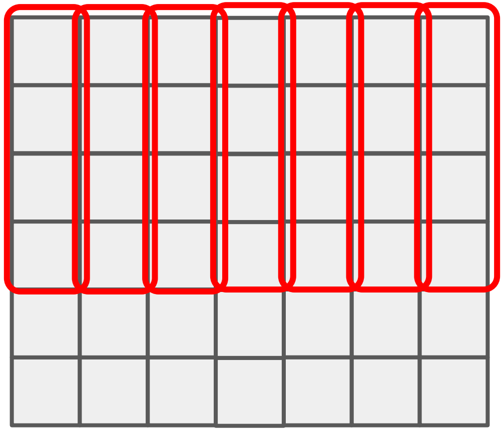


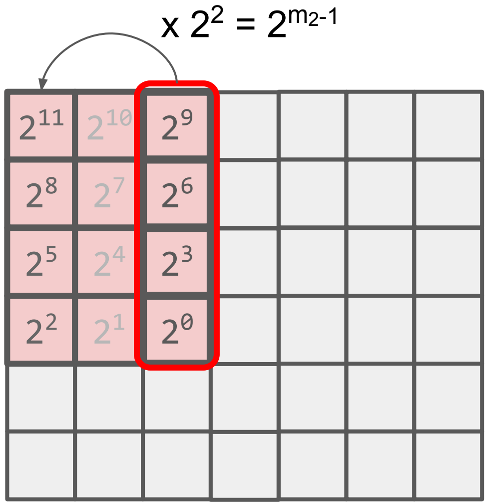


Fleshing it out

How should we hash the vertical columns?

Look at the arrangement of the **full pattern** -- take the rightmost column. It is the "simplest". Also easily extends to the leftmost column!

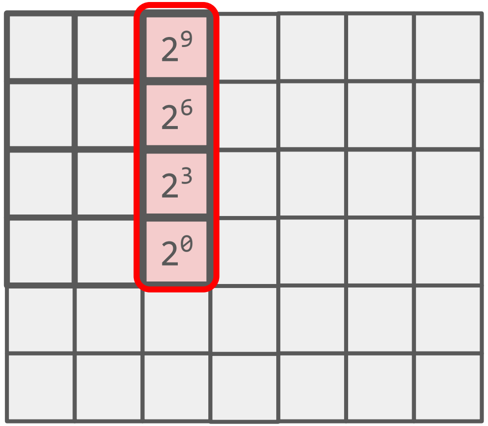




Fleshing it out (column hashes)

Let $C_{i,j}$ be the column hashes starting from index (i, j) as the topmost:

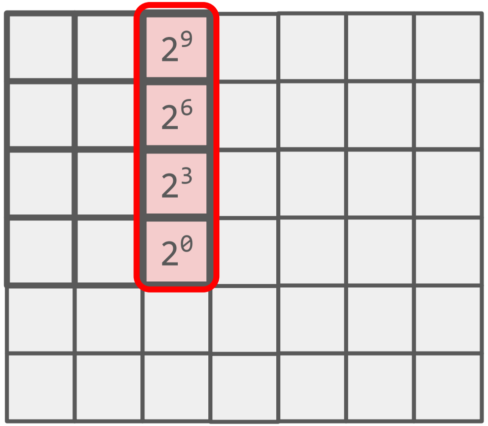
e.g.
The part of text corresponding to $C_{1,3}$



Fleshing it out (column hashes)

Let $C_{i,j}$ be the column hashes starting from index (i, j) as the topmost:

e.g.
The part of text corresponding to $C_{1,3}$



$$C_{i,j} = \sum_{k=0}^{m_1-1} h_p(2^{km_2}) \cdot T[i + m_1 - 1 - k, j] \pmod{p}$$

Looks scary, but this is just the generalised form of what we have on the left!

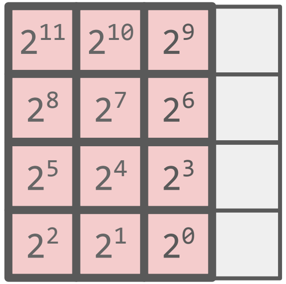
We are ensuring that the powers skip appropriately!

Note: This is just the idea, there are sloppy details

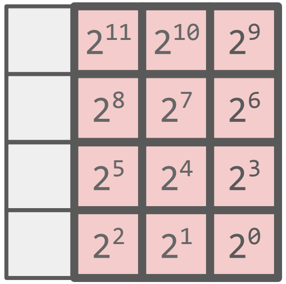
$$C_{i,j} = \sum_{k=0}^{m_1-1} h_p(2^{km_2}) \cdot T[i+m_1-1-k, j] \pmod p$$

Fleshing it out (rolling hash by column)

Call this region $h(y_1)$



Call this region $h(y_2)$

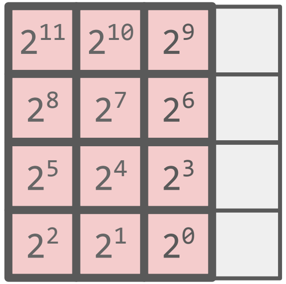


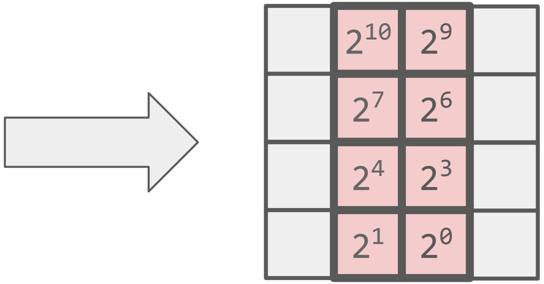
Note: This is just the idea, there are sloppy details

$$C_{i,j} = \sum_{k=0}^{m_1-1} h_p(2^{km_2}) \cdot T[i+m_1-1-k, j] \pmod p$$

Fleshing it out (rolling hash by column)

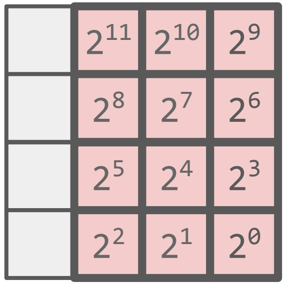
Call this region $h(y_1)$





$h(y_1) - 2^2 \cdot C_{i,j}$

Call this region $h(y_2)$



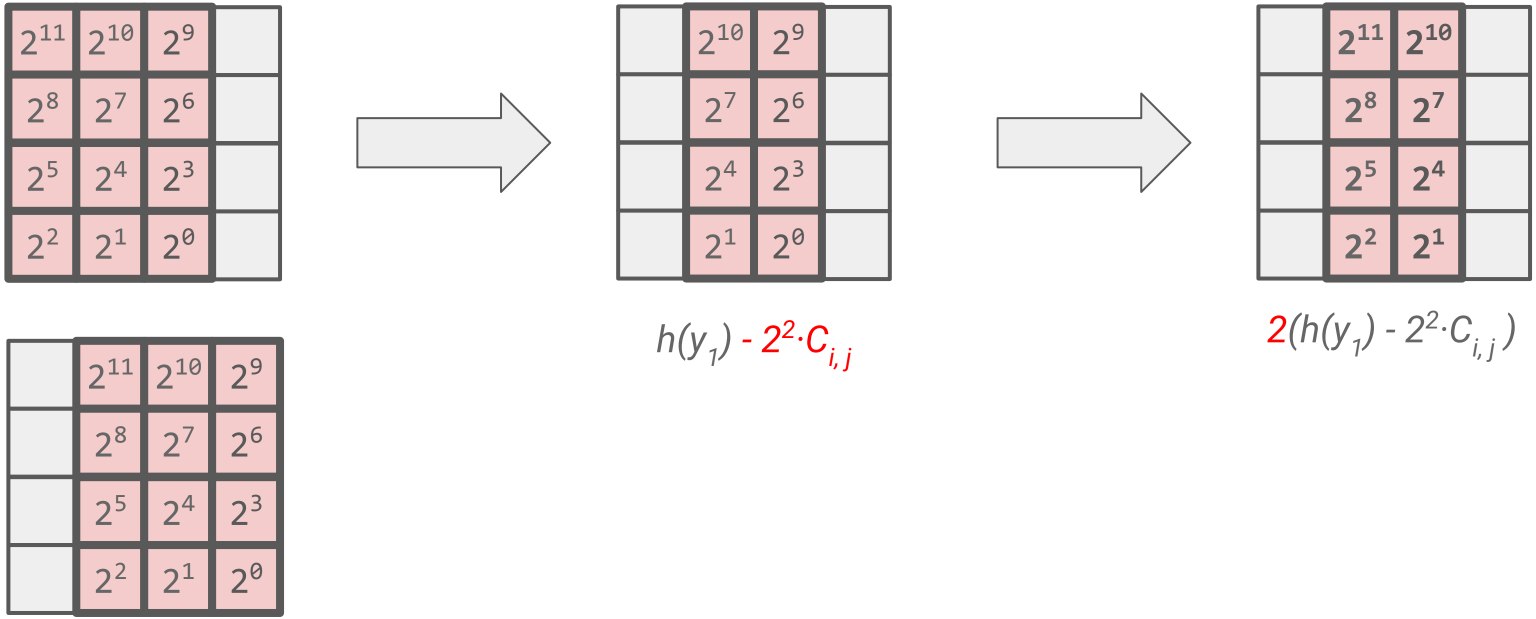
Note: This is just the idea, there are sloppy details

$$C_{i,j} = \sum_{k=0}^{m_1 - 1} h_p(2^{km_2}) \cdot T[i + m_1 - 1 - k, j] \pmod p$$

Fleshing it out (rolling hash by column)

Call this region $h(y_1)$

Call this region $h(y_2)$



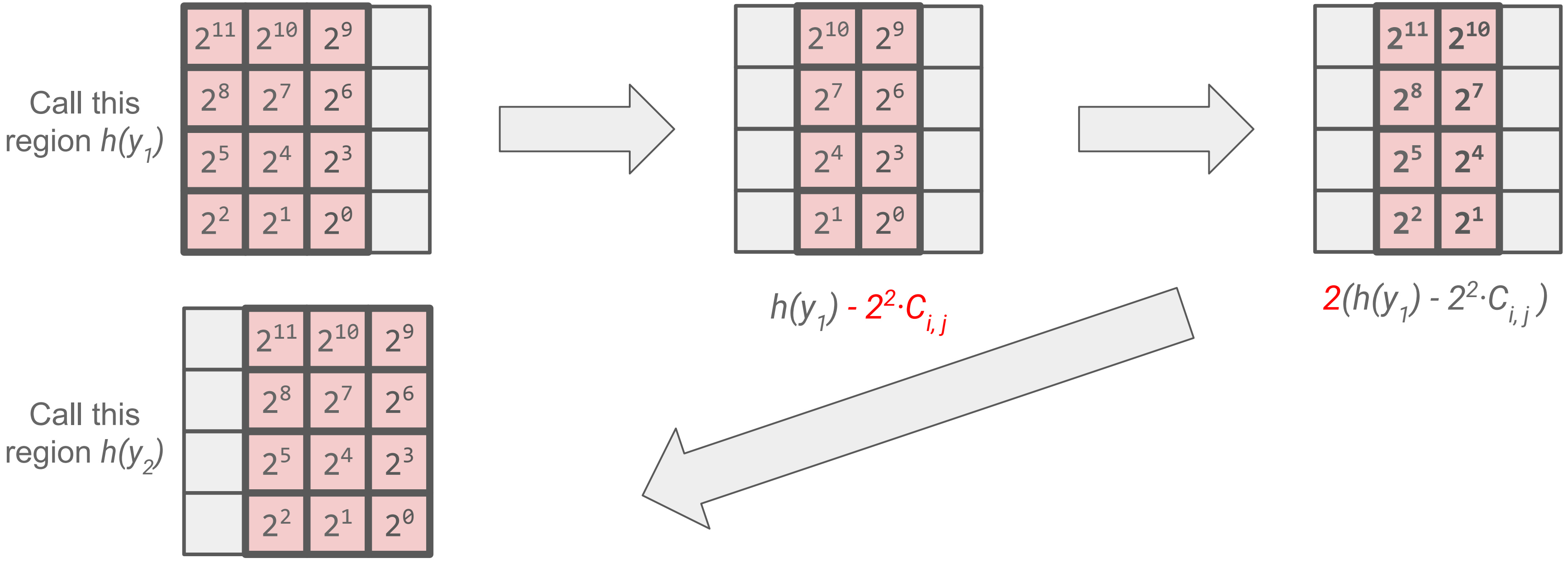
$h(y_1) - 2^2 \cdot C_{i,j}$

$2(h(y_1) - 2^2 \cdot C_{i,j})$

Note: This is just the idea, there are sloppy details

$$C_{i,j} = \sum_{k=0}^{m_1-1} h_p(2^{km_2}) \cdot T[i+m_1-1-k, j] \pmod p$$

Fleshing it out (rolling hash by column)



$2(h(y_1) - 2^2 \cdot C_{i,j}) + C_{i,j+3} = 2h(y_1) - 2^3 \cdot C_{i,j} + C_{i,j+3}$

$$C_{i,j} = \sum_{k=0}^{m_1-1} h_p(2^{km_2}) \cdot T[i+m_1-1-k, j] \pmod p$$

Fleshing it out (rolling hash by column)

If R is a subrectangle with northwest corner at *(i, j)* and R' at *(i, j+1):*

$$C_{i,j} = \sum_{k=0}^{m_1-1} h_p(2^{km_2}) \cdot T[i+m_1-1-k, j] \pmod{p}$$

Fleshing it out (rolling hash by column)

If R is a subrectangle with northwest corner at *(i, j)* and R' at *(i, j+1)*:

$$h_p(R') = 2h_p(R) - h_p(2^{m_2}) \cdot C_{i,j} + C_{i,j+m_2} \pmod{p}$$

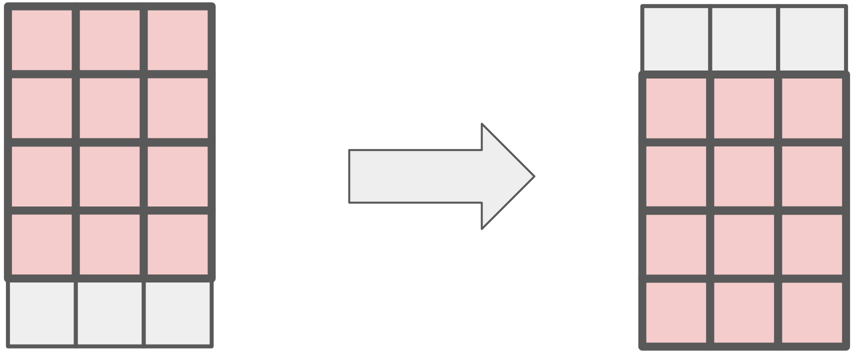
Call this region $h(y_1)$

Call this region $h(y_2)$

$2h(y_1) - 2^3 \cdot C_{i,j} + C_{i,j+3}$

Fleshing it out (rolling hash by row)

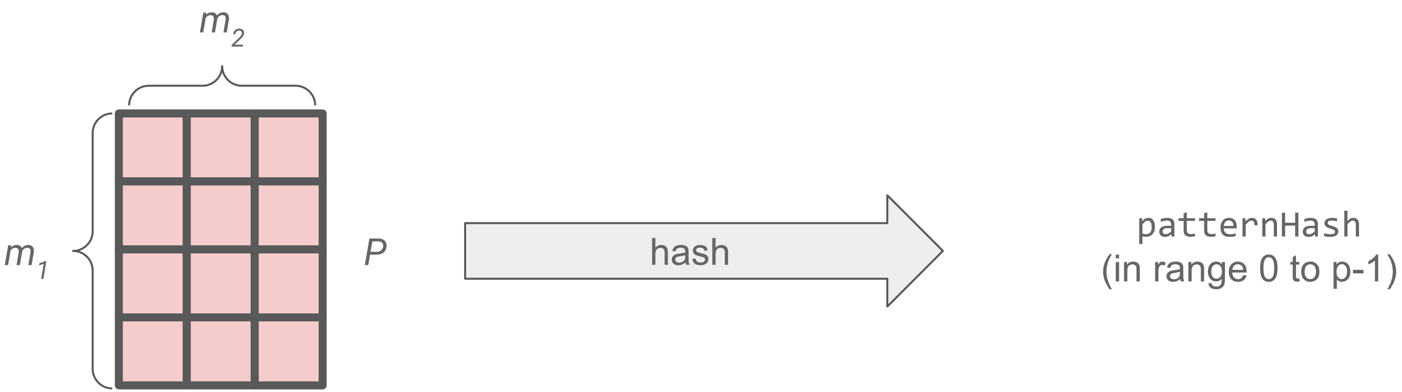
Computing the hash values for row, and to “roll down” can be done in a similar manner:



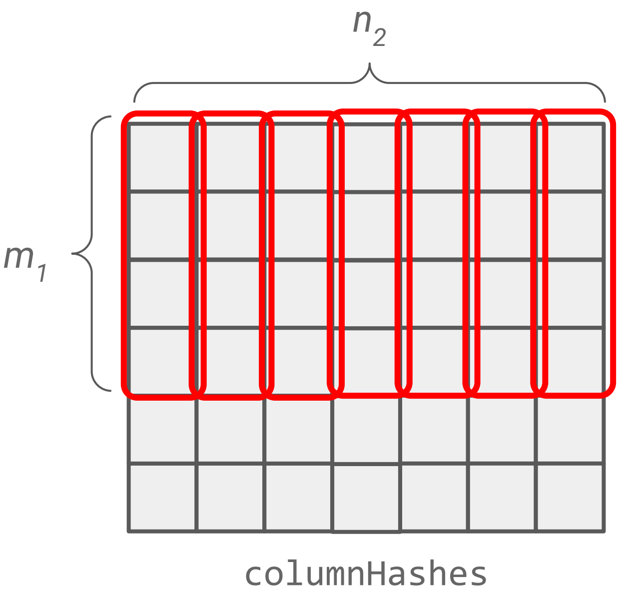
Pseudocode

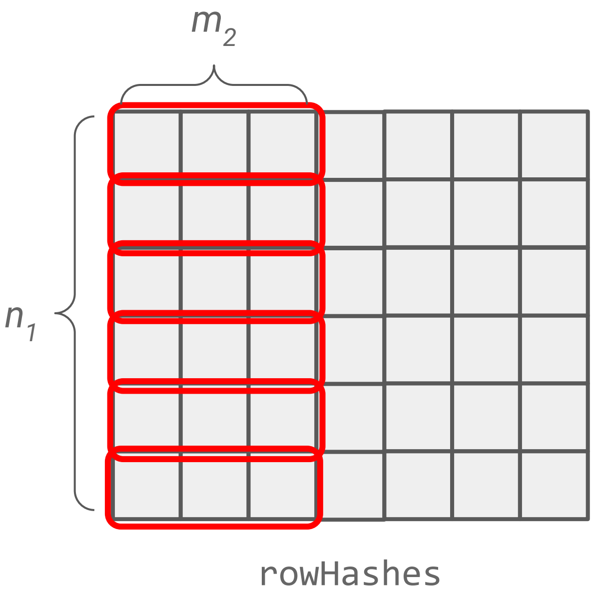
```
def 2D-karp-rabin(T, P):
patternHash = hash P
columnHashes = find all n2 column hashes of the first m1 rows
rowHashes = find all n1 row hashes of the first m2 columns
textSubrectangleHash = hash the m1 x m2 subrectangle on the northwest corner
textSubrectangleHashes = apply rowHashes to get all the m1 x m2 first hashes
for r in 1 to n1-m1+1:
    match and roll textSubrectangleHashes[r] horizontally using columnHashes
    return True if there is a match
    for c in in 1 to n2:
        roll columnHashes[c] vertically down
return False
```

```
def 2D-karp-rabin(T, P):
01. patternHash = hash P
```

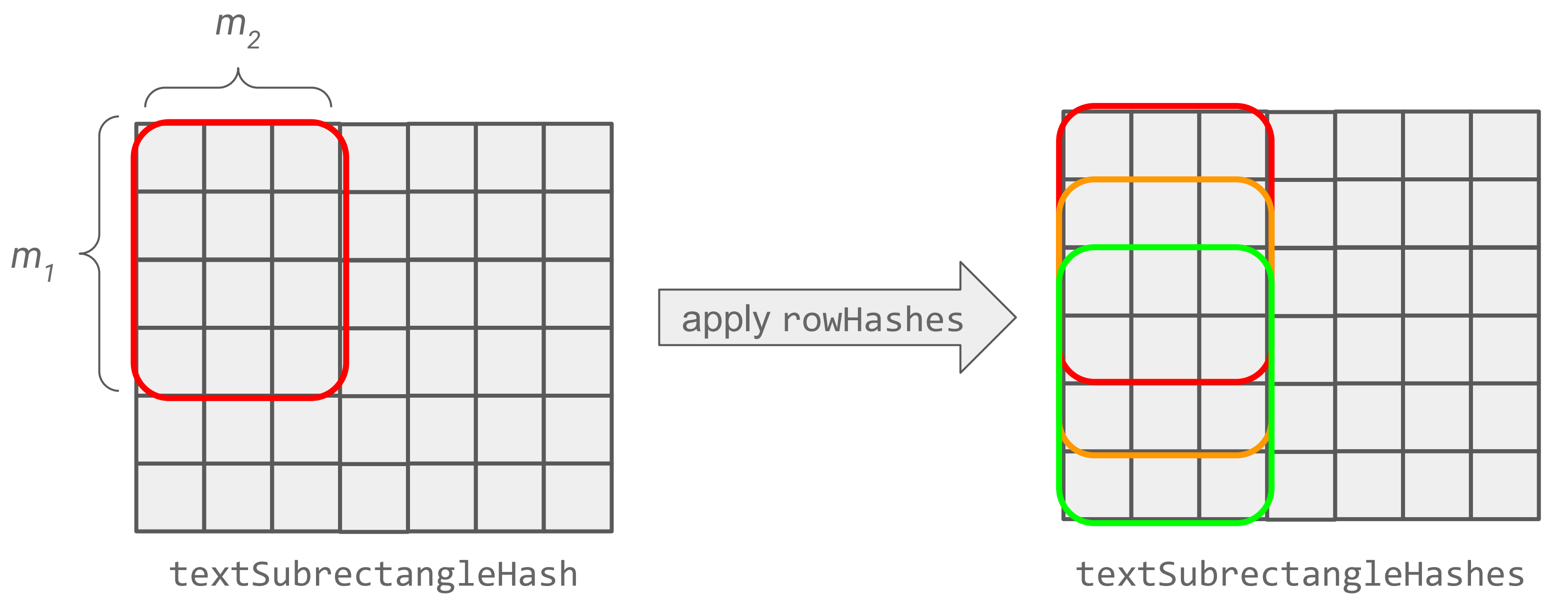


```
02. columnHashes = find all n2 column hashes of the first m1 rows
03. rowHashes = find all n1 row hashes of the first m2 columns
```

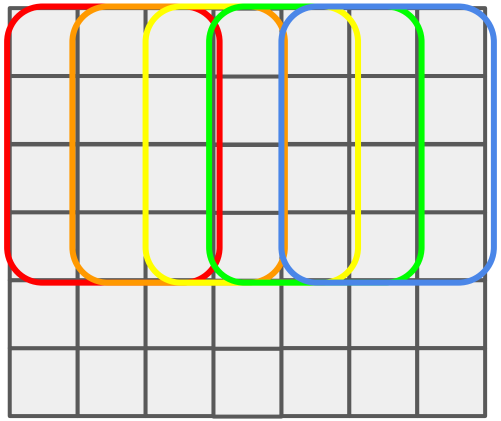




```
04. textSubrectangleHash = hash the m1 x m2 subrectangle on the northwest corner
05. textSubrectangleHashes = apply rowHashes to get all the m1 x m2 first hashes
```

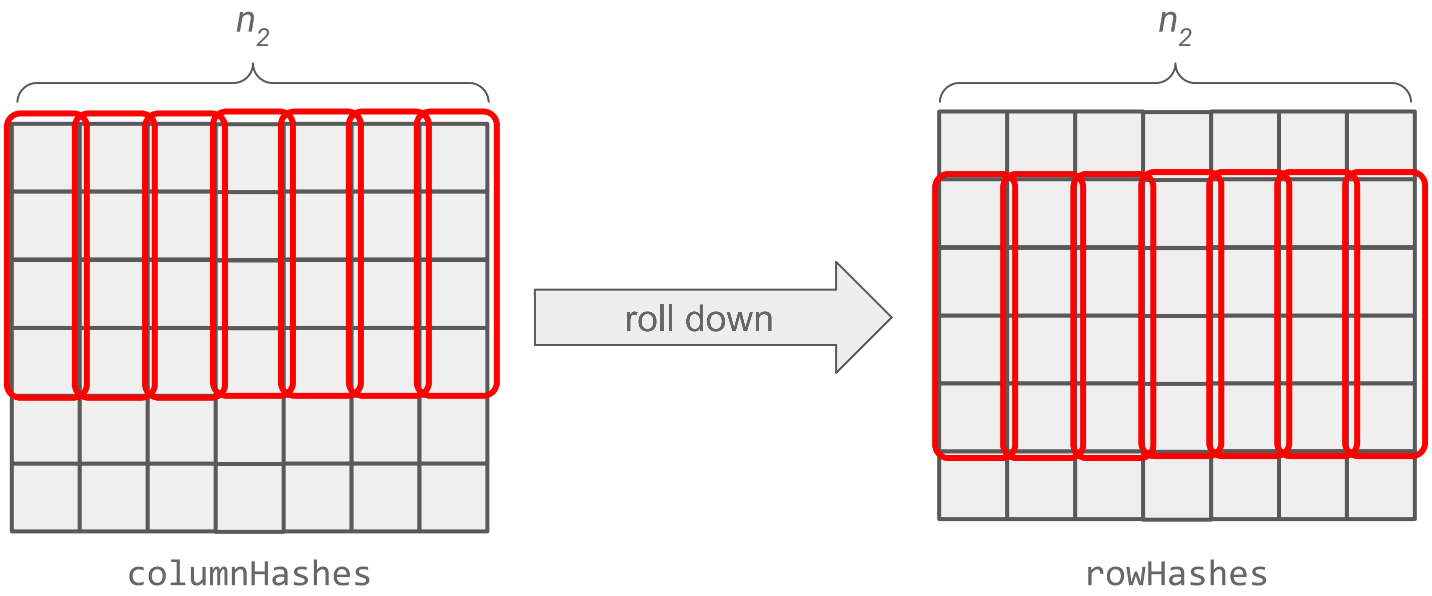


```
for r in 1 to n1-m1+1:
    match and roll textSubrectangleHashes[r] horizontally using columnHashes
    return True if there is a match
    for c in in 1 to n2:
        roll columnHashes[c] vertically down
```

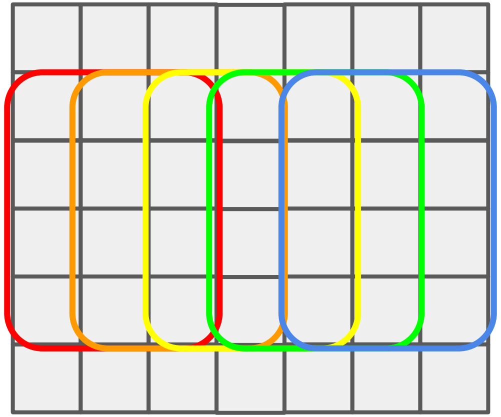


Applying `columnHashes` to roll horizontally right

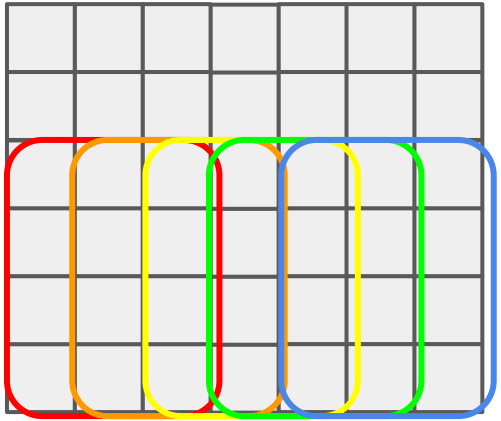
```
for r in 1 to n1-m1+1:
    match and roll textSubrectangleHashes[r] horizontally using columnHashes
    return True if there is a match
    for c in in 1 to n2:
        roll columnHashes[c] vertically down
```



```
for r in 1 to n1-m1+1:
    match and roll textSubrectangleHashes[r] horizontally using columnHashes
    return True if there is a match
    for c in in 1 to n2:
        roll columnHashes[c] vertically down
```



Try the remaining rows

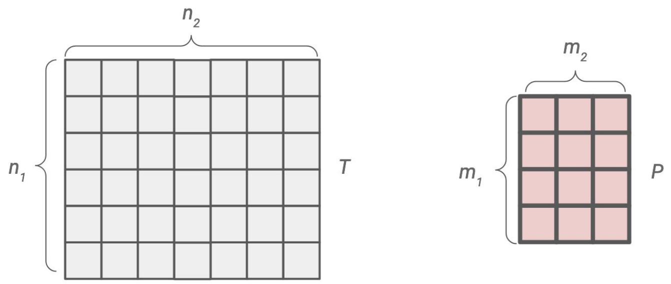


Runtime Analysis (Goal: $O(n_1 n_2)$)

```
def 2D-karp-rabin(T, P):
01. patternHash = hash P
02. columnHashes = find all n2 column hashes of the first m1 rows
03. rowHashes = find all n1 row hashes of the first m2 columns
04. textSubrectangleHash = hash the m1 x m2 subrectangle on the northwest corner
05. textSubrectangleHashes = apply rowHashes to get all the m1 x m2 first hashes
```

Line 1: $O(m_1 m_2)$
Line 2: $O(m_1 n_2)$
Line 3: $O(n_1 m_2)$
Line 4: $O(m_1 m_2)$
Line 5: $O(n_1)$ -- use $O(1)$ rolling hash down the n_1 text

All $O(n_1 n_2)$. So far so good!



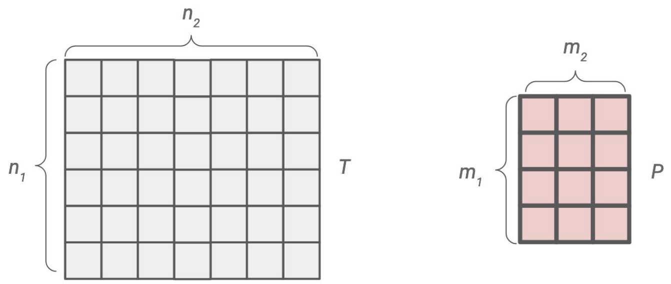
Runtime Analysis (Goal: $O(n_1 n_2)$)

```
06. for r in 1 to n1-m1+1:
07.     match and roll textSubrectangleHashes[r] horizontally using columnHashes
08.     return True if there is a match
09.     for c in in 1 to n2:
10.         roll columnHashes[c] vertically down
11. return False
```

Loop body:

Line 7: $O(n_2)$ - via $O(1)$ rolling hash down the n_2 columns

Line 9: $O(n_2)$ - also $O(1)$ rolling hash



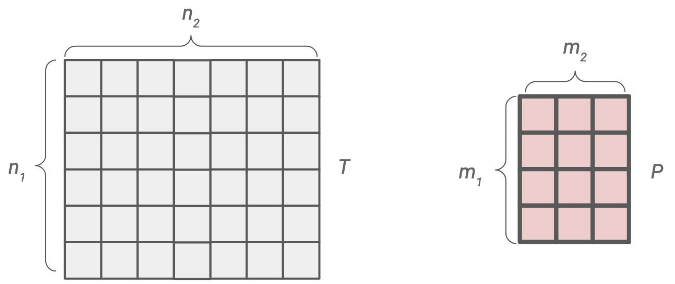
Runtime Analysis (Goal: $O(n_1 n_2)$)

```
for r in 1 to n1-m1+1:
    match and roll textSubrectangleHashes[r] horizontally using columnHashes
    return True if there is a match
    for c in in 1 to n2:
        roll columnHashes[c] vertically down
return False
```

Loop body: $O(n_2)$

Looped for n_1-m_1+1 times = $O(n_1)$

Total time: $O(n_1 n_2)$



Correctness Analysis (Goal: False Positive < 1%)

```
for r in 1 to n1-m1+1:
    match and roll textSubrectangleHashes[r] horizontally using columnHashes
    return True if there is a match
    for c in in 1 to n2:
        roll columnHashes[c] vertically down
return False
```

We haven’t set the range of prime numbers yet! Work backwards to figure out the number.

Correctness Analysis (Goal: False Positive < 1%)

```
06. for r in 1 to n1-m1+1:
07.     match and roll textSubrectangleHashes[r] horizontally using columnHashes
08.     return True if there is a match
09.     for c in in 1 to n2:
10.         roll columnHashes[c] vertically down
11. return False
```

We haven't set the range of prime numbers yet! Work backwards to figure out the number.

Every time line 7 is called, it is possibly a false positive. This line is called $(n_1-m_1+1)(n_2-m_2+1) = O(n_1n_2)$ times

Correctness Analysis (Goal: False Positive < 1%)

Recall: Union Bound

$$\Pr[A \text{ or } B] \leq \Pr[A] + \Pr[B]$$

Correctness Analysis (Goal: False Positive < 1%)

Recall: Union Bound

$$\Pr[A \text{ or } B] \leq \Pr[A] + \Pr[B]$$

Let $\mathcal{E}_i$ denote the event that the i-th match returns a false positive

$$\Pr[\mathcal{E}_1 \vee \mathcal{E}_2 \vee \cdots \vee \mathcal{E}_{n_1 n_2}] \leq \Pr[\mathcal{E}_1] + \Pr[\mathcal{E}_2] + \cdots + \Pr[\mathcal{E}_{n_1 n_2}]$$

$$\boxed{= \frac{1}{100}}$$

Ideally, this is our goal!

Correctness Analysis (Goal: False Positive < 1%)

Recall: Union Bound

$$\Pr[A \text{ or } B] \leq \Pr[A] + \Pr[B]$$

Let $\mathcal{E}_i$ denote the event that the i-th match returns a false positive

$$\Pr[\mathcal{E}_1 \vee \mathcal{E}_2 \vee \cdots \vee \mathcal{E}_{n_1 n_2}] \leq \Pr[\mathcal{E}_1] + \Pr[\mathcal{E}_2] + \cdots + \boxed{\Pr[\mathcal{E}_{n_1 n_2}]}$$

$$= \frac{1}{100}$$

To do that, each one of these should be

$$\frac{1}{100 n_1 n_2}$$

Correctness Analysis

Claim: If $0 \leq x < y < \boxed{2^b}$, then:

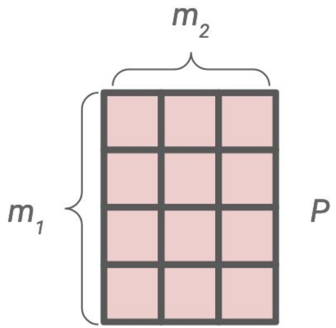
$$\Pr_p[h_p(x) = h_p(y)] < \frac{b \ln K}{K}.$$

Let $\mathcal{E}_i$ denote the event that the i-th match returns a false positive

P = pattern, R = rectangle associated with the i^th match

$$\Pr[\mathcal{E}_i] = \Pr[P \neq R \text{ but } h_p(P) = h_p(R)]$$
$$\leq \frac{\boxed{m_1 m_2} \lg(K)}{K}$$

We view the pattern as an integer with length $m_1 m_2$ bits



Correctness Analysis

Claim: If $0 \le x < y < \boxed{2^b,}$ then:

$$\Pr_p[h_p(x) = h_p(y)] < \frac{b \ln K}{K}.$$

Let $\mathcal{E}_i$ denote the event that the i-th match returns a false positive

P = pattern, R = rectangle associated with the i^{th} match

$$\Pr[\mathcal{E}_i] = \Pr[P \neq R \text{ but } h_p(P) = h_p(R)]$$

$$\le \frac{\boxed{m_1 m_2} \lg(K)}{K}$$

Similar analysis as before and in lecture

Set $K = \Theta(n_1 n_2 m_1 m_2 \lg(n_1 n_2 m_1 m_2))$ and we will obtain $\Pr[\mathcal{E}_i] \le \frac{1}{100 n_1 n_2}$

Furthermore, we can fit K with O(lgK) = O(lg n_1 + lg n_2) bits → fits in constant number of machine words in the Word-RAM model

Correctness Analysis (Goal: False Positive < 1%)

Thus we will have this as desired!

$$\begin{aligned}\Pr[\mathcal{E}_1 \vee \mathcal{E}_2 \vee \cdots \vee \mathcal{E}_{n_1 n_2}] &\leq \Pr[\mathcal{E}_1] + \Pr[\mathcal{E}_2] + \cdots + \Pr[\mathcal{E}_{n_1 n_2}] \\ &= \frac{1}{100}\end{aligned}$$