2 February 2025

Dynamic Programming II

Edit Distance

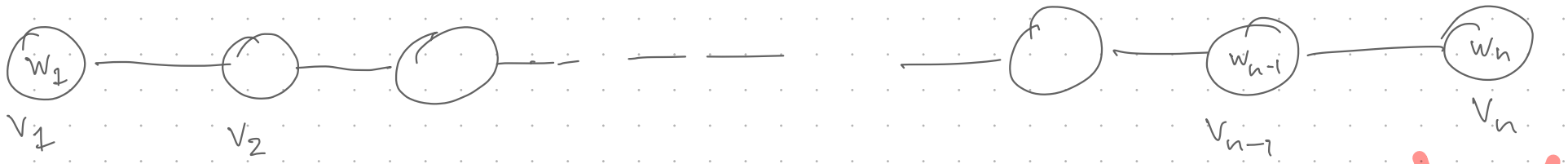
Plan

* Review of Dynamic Programming
* Announcements
* Sequence Alignment Problem (aka Edit Distance)

Dynamic Programming

* Search for optimal solution by exploiting sub-structure.

Example. Max Wt Independent Set on Paths



$v_n \in S$ OR $v_n \notin S$

$$WIS(P_n) = \max \begin{cases} WIS(P_{n-1}), \\ w_n + WIS(P_{n-2}) \end{cases}$$

DP Recurrence $\Longrightarrow$ Recursive Algorithm

Key Ingredient: Record Answers as you go!

↳ Avoid Redoing work.

$W = [-1, -1, \ldots, -1]$ // Dynamic Programming Table

Memoized WIS (P_k).

if $k=0$, return 0
if $k=1$, return w_1.

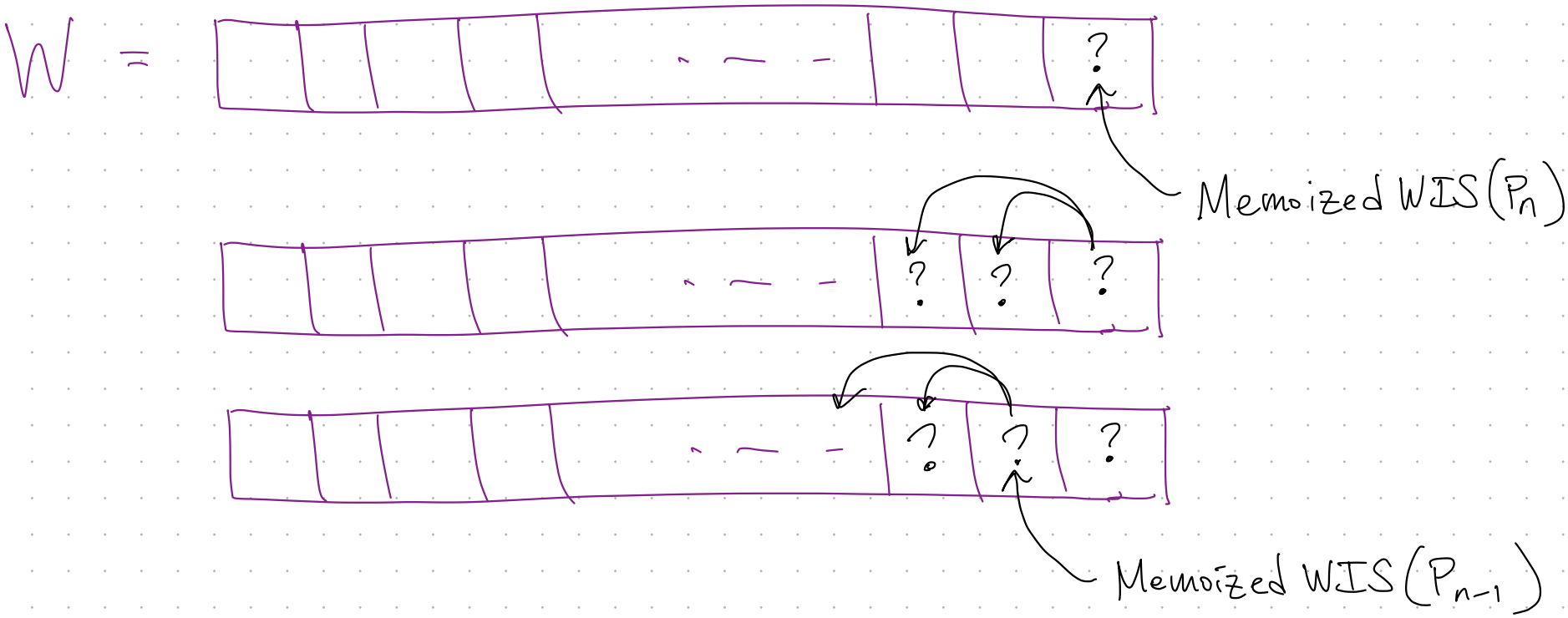
if $W[k-1] = -1$:

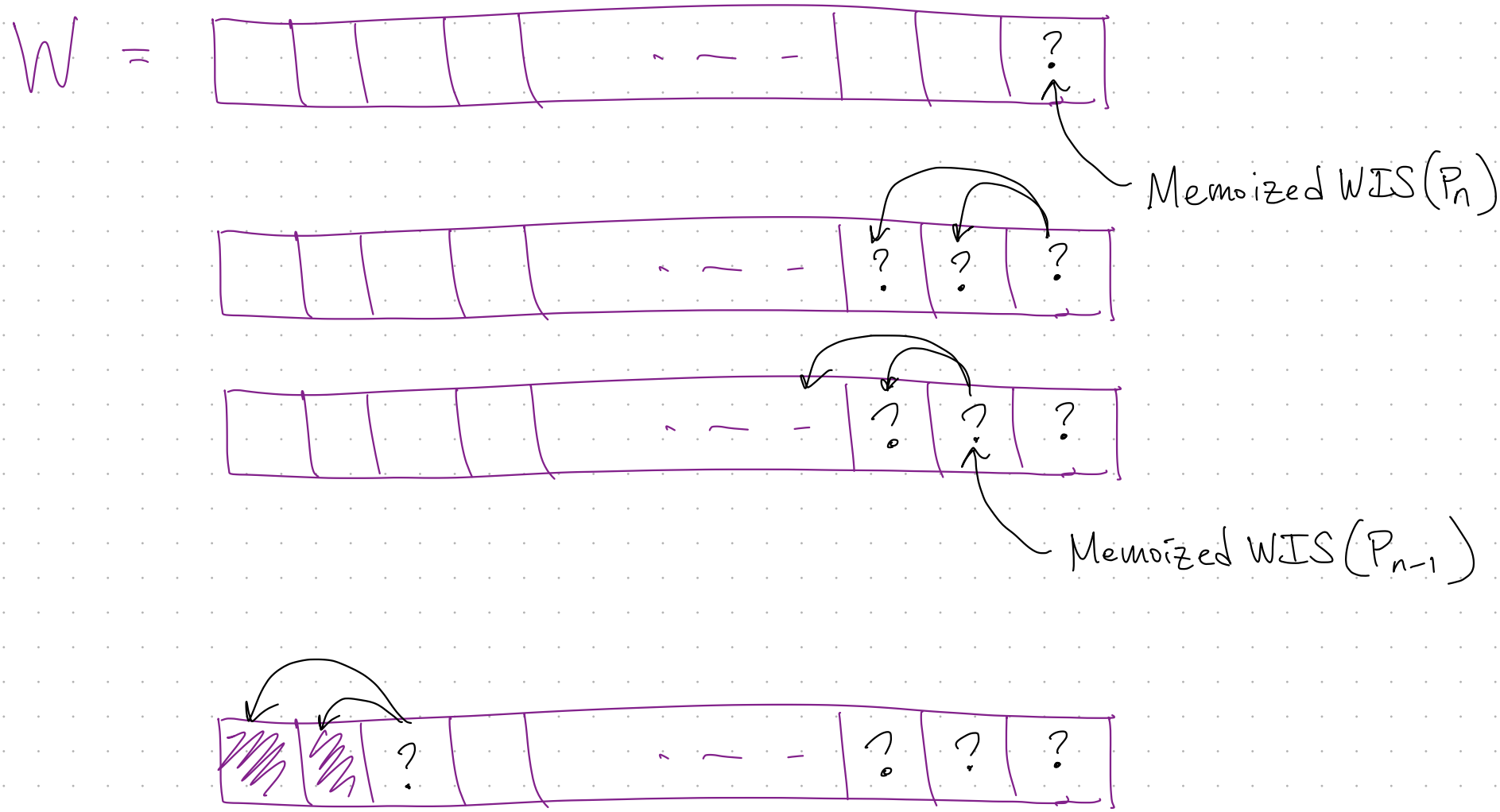
    $W[k-1] \leftarrow$ Memoized WIS (P_{k-1})

if $W[k-2] = -1$:

    $W[k-2] \leftarrow$ Memoized WIS (P_{k-2})

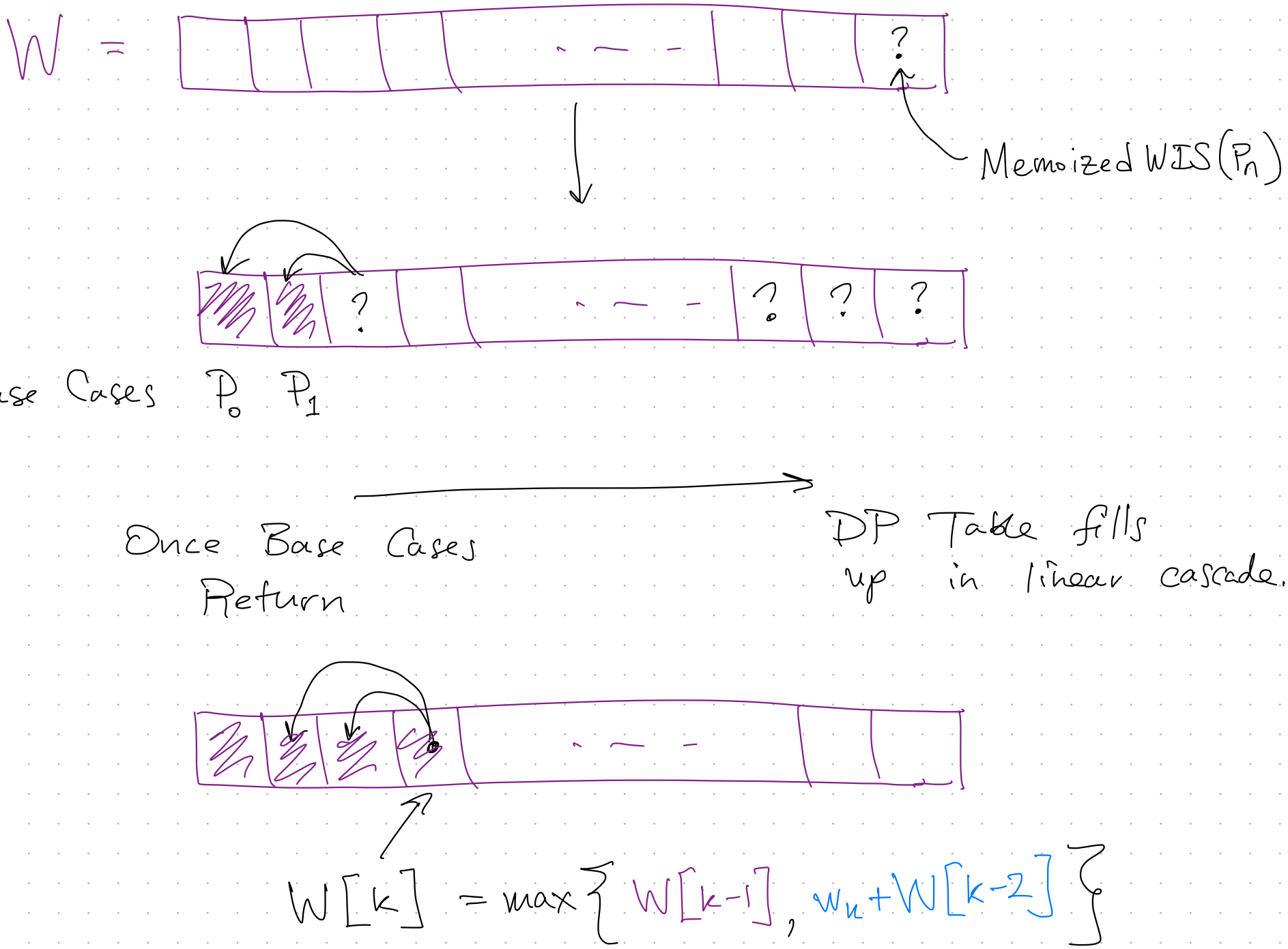
return $\max\{W[k-1], w_k + W[k-2]\}$


W =
?
Memoized WIS(P_n)
? ? ?
? ? ?
Memoized WIS(P_{n-1})



Once Base Cases Return

DP Table fills up in linear cascade.


W =
?
Memoized WIS(P_n)
?
?
?
?
Base Cases P_0 P_1
Once Base Cases Return
DP Table fills up in linear cascade.
$W[k] = \max\{ W[k-1], w_k + W[k-2] \}$

Iterative Solution.

* Recursive formulation conceptually nice.
* Dynamic Programs always have an equivalent iterative formulation

  ↳ fills in the DP Table directly

Iterative Solution.

* Recursive formulation conceptually nice.
* Dynamic Programs always have an equivalent iterative formulation

↳ fills in the DP Table directly

From here on, we write the iterative versions

Iterative WIS (P_n).

Let $W = [-1, -1, \ldots, -1]$

$W[0] \leftarrow 0$, $W[1] = w_1$.

for $k = 2, \ldots, n$:

$W[k] \leftarrow \max \{ W[k-1], w_k + W[k-2] \}$

Return $W[n]$

Announcements.

* HW0 Grades Released After lecture
* HW1
* Exam conflict Survey
  - ↳ Posted to Ed.
  - ↳ Due Thurs.
  - ↳ Only for University-approved conflicts

Proofs of Correctness.

* What does it mean to prove algorithm A is "correct" for problem Π?

For all instances x of problem Π,
$A(x)$ returns the right answer.

Proofs of Correctness.

* What does it mean to prove algorithm A is "correct" for problem Π?

> For all instances x of problem Π,
> $A(x)$ returns the right answer.

* Be precise & Break Correctness in pieces.

↳ e.g. $\forall$ connected undirected weighted graphs G, w/ distinct edge weights
Kruskal's returns MST of G.

Show. Kruskal's returns
- a tree T
- T is spanning (ie. connected)
- T is minimum wt

Dynamic Programming Proofs of Correctness

* Need to prove the Recurrence is correct.

Ex. For all $n \in \mathbb{N}$ and any non-negative vertex weights on the path graph P_n,

$$WIS(P_n) = \max\{P_{n-1}, w_n + P_{n-2}\}$$

OR

Base Cases

Sequence Alignment.

Genomics

* Genome Sequences $S \in \{A, C, G, T\}^*$

$S_{cat} = A\ C\ C\ G\ A\ T\ C\ G\ A\ T \ldots$

$S_{dog} = A\ C\ G\ G\ A\ A\ T\ C\ G\ G\ T \ldots$

How related are two species?

↳ How "similar" are their genomes?

Genomics

* Genome Sequences $S \in \{A, C, G, T\}^*$

$S_{cat} =$ A C C G A T C G A T ...

$S_{dog} =$ A C G G A A T C G G T ....

How related are two species?

↳ How "similar" are their genomes?

BLAST: Basic Local Alignment Search Tool >110K Citations

Edit Distance

How easily can we edit string S into string t?

Edit Distance

How easily can we edit string S into string t?

Allowed Edits

* Insertion / Deletion
* Changes

EDIT MINIMUM SNOW

DIST MAXIMUM NO

Edit Distance

How easily can we edit string S into string t?

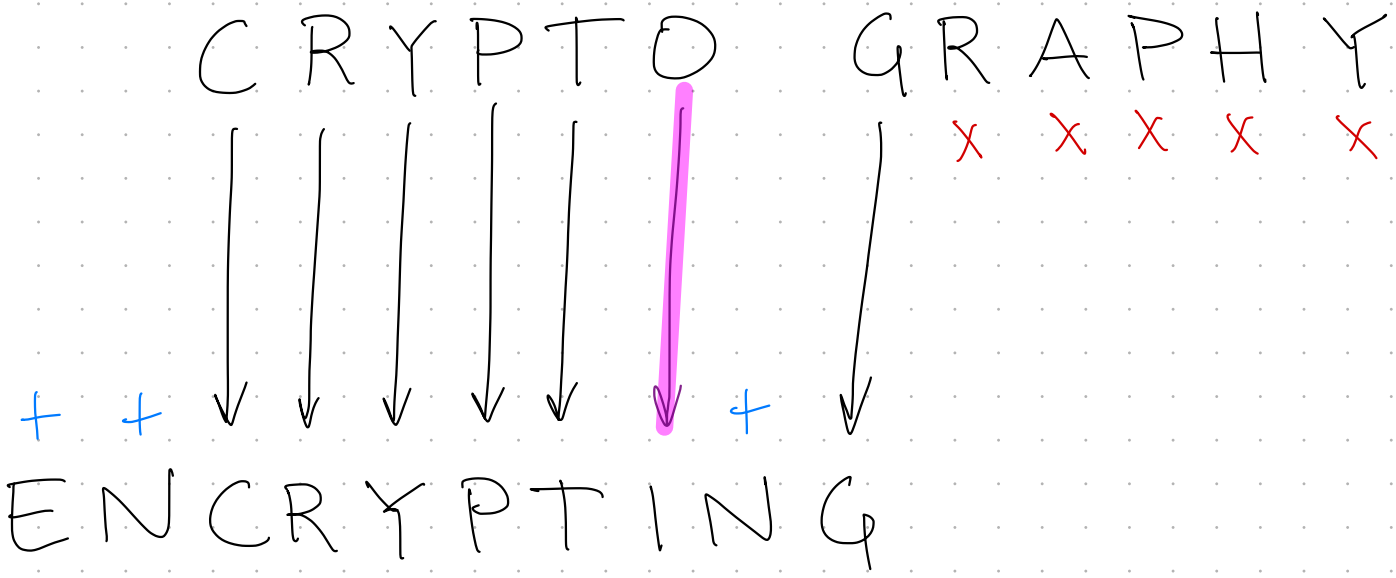
CRYPTOGRAPHY

ENCRYPTING

?

Edit Distance

How easily can we edit string S into string t?



3 insertions

5 deletion

1 edit

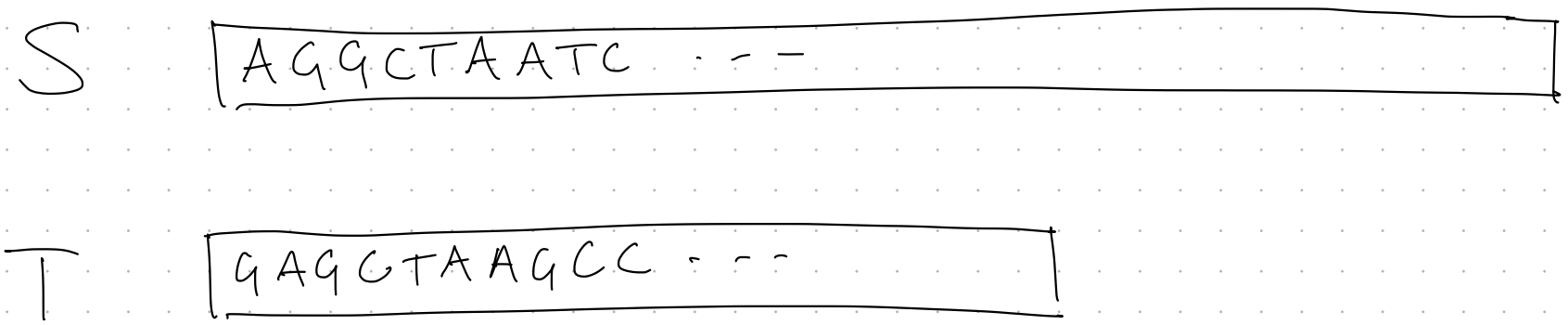
O → I

Given two strings S and T

Compute minimum cost edits from $S \rightarrow T$

insertion / deletion: γ

"changing" a to b: Δ_{ab}



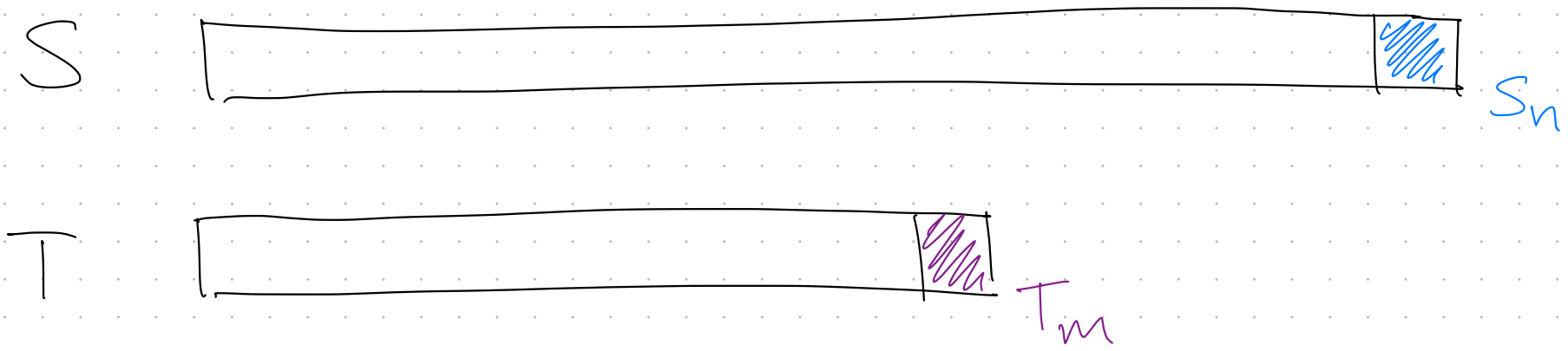
Changing a to a costs $\Delta_{aa} = 0$.

Given two strings S and T

Compute minimum cost edits from $S \rightarrow T$

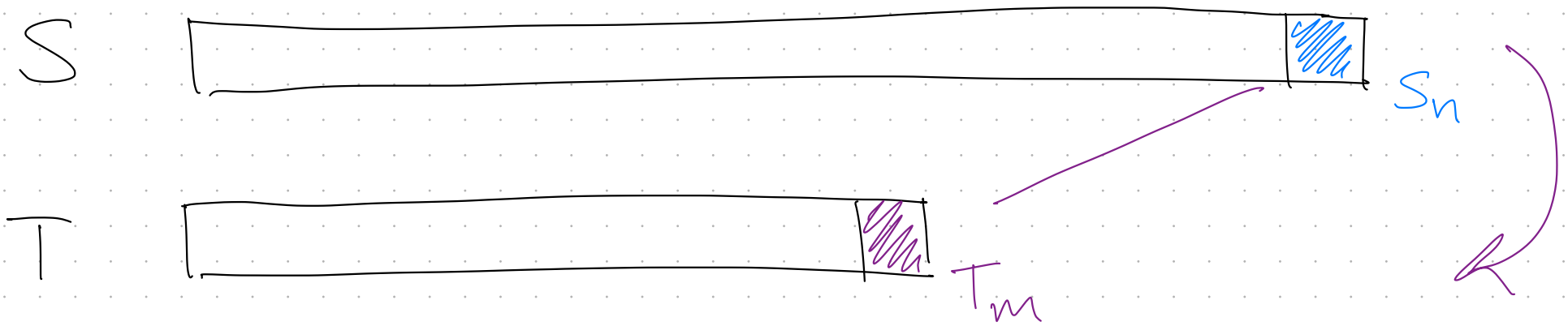
insertion / deletion: γ

changing a to b: Δ_{ab}



Fact. Fix an optimal set of edits from S to T.

One of the following is true:

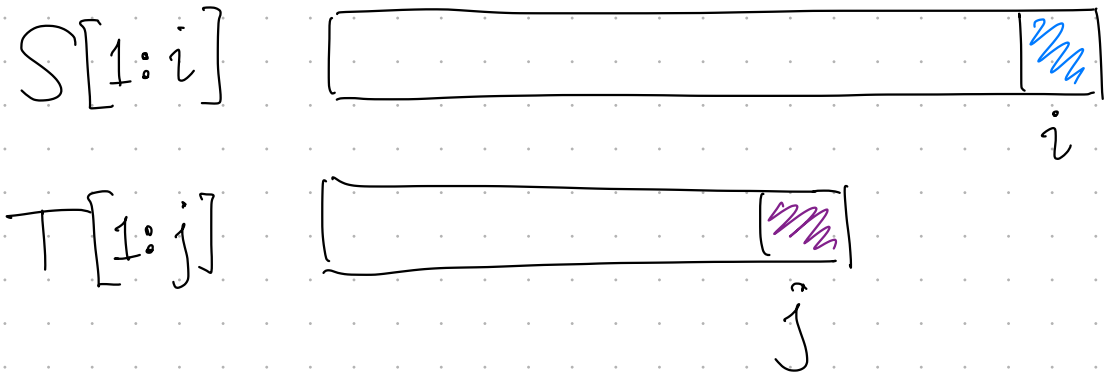


Fact. Fix an optimal set of edits from S to T.

One of the following is true:

* S_n is deleted at cost γ
* T_m is inserted at cost γ
* S_n "changes" to T_m at cost $\Delta_{S_n T_m}$

The Edit Distance Recurrence

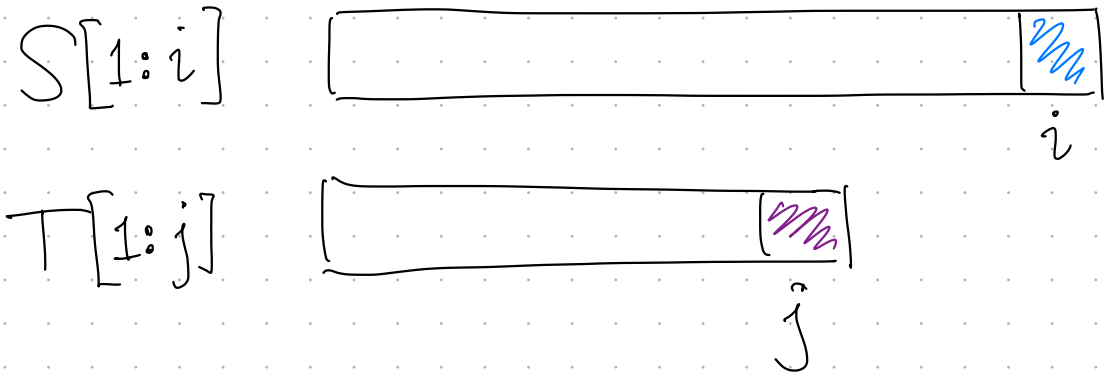


$ED(i,j)$

↓

Edit distance between prefixes $S[1:i]$ and $T[1:j]$

The Edit Distance Recurrence



$ED(i,j)$

Possibilities

* S_i deleted $\longrightarrow$ $ED(i-1,j) + \gamma$
* T_j inserted $\longrightarrow$ $ED(i,j-1) + \gamma$
* S_i changes to T_j $\longrightarrow$ $ED(i-1,j-1) + \Delta_{S_i T_j}$

Theorem. The Edit Distance between $S[1:i]$ and $T[1:j]$ is given as

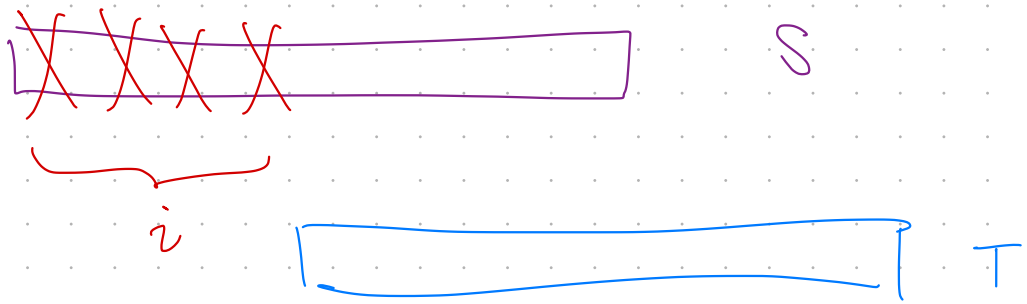
$$ED(i,j) = \min \begin{cases} ED(i-1, j-1) + \Delta_{S_i T_j}, \\ ED(i-1, j) + \gamma, \\ ED(i, j-1) + \gamma \end{cases}$$

Theorem. The Edit Distance between $S[1:i]$ and $T[1:j]$ is given as

$$ED(i,j) = \min \begin{cases} ED(i-1, j-1) + \Delta_{S_i T_j}, \\ ED(i-1, j) + \gamma, \\ ED(i, j-1) + \gamma \end{cases}$$

Base Cases?

$$ED(i, 0) = i \cdot \gamma \longrightarrow$$



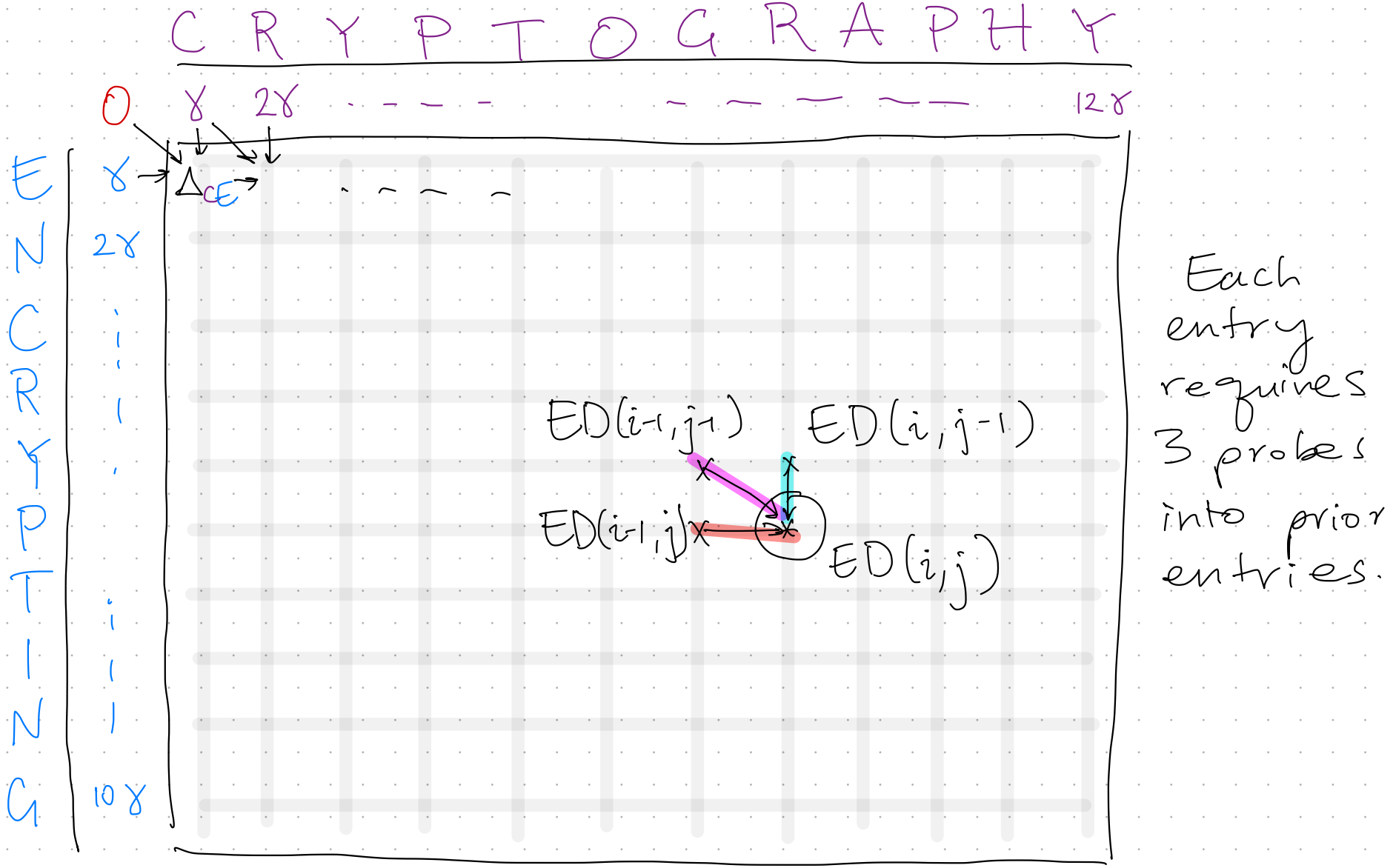
$$ED(0, j) = j \cdot \gamma$$

Theorem. The Edit Distance between $S[1:i]$ and $T[1:j]$ is given as

$$ED(i,j) = \min \begin{cases} ED(i-1, j-1) + \Delta_{S_i T_j}, \\ ED(i-1, j) + \gamma, \\ ED(i, j-1) + \gamma \end{cases}$$

$ED(i,j)$

2D dynamic programming table



Each entry requires 3 probes into prior entries.

Edit Distance Algorithm

$ED(0,0) = 0$

$ED(i,0) = i \cdot \gamma$

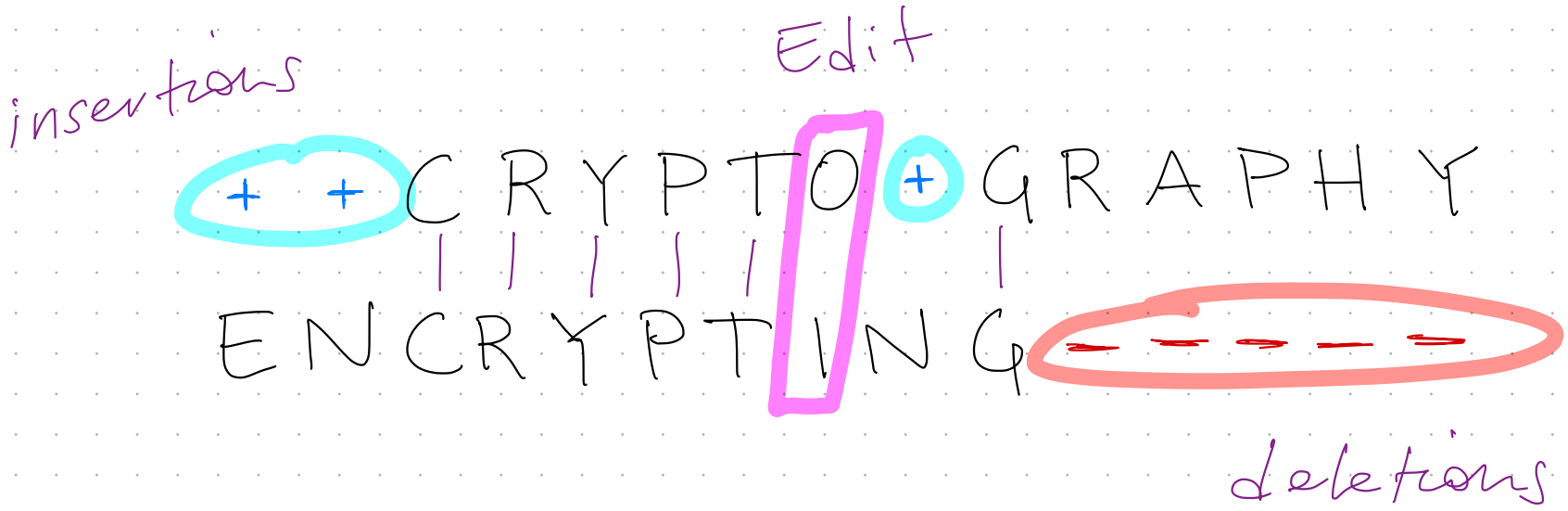
$ED(0,j) = j \cdot \gamma$

For $i = 1 \longrightarrow n$

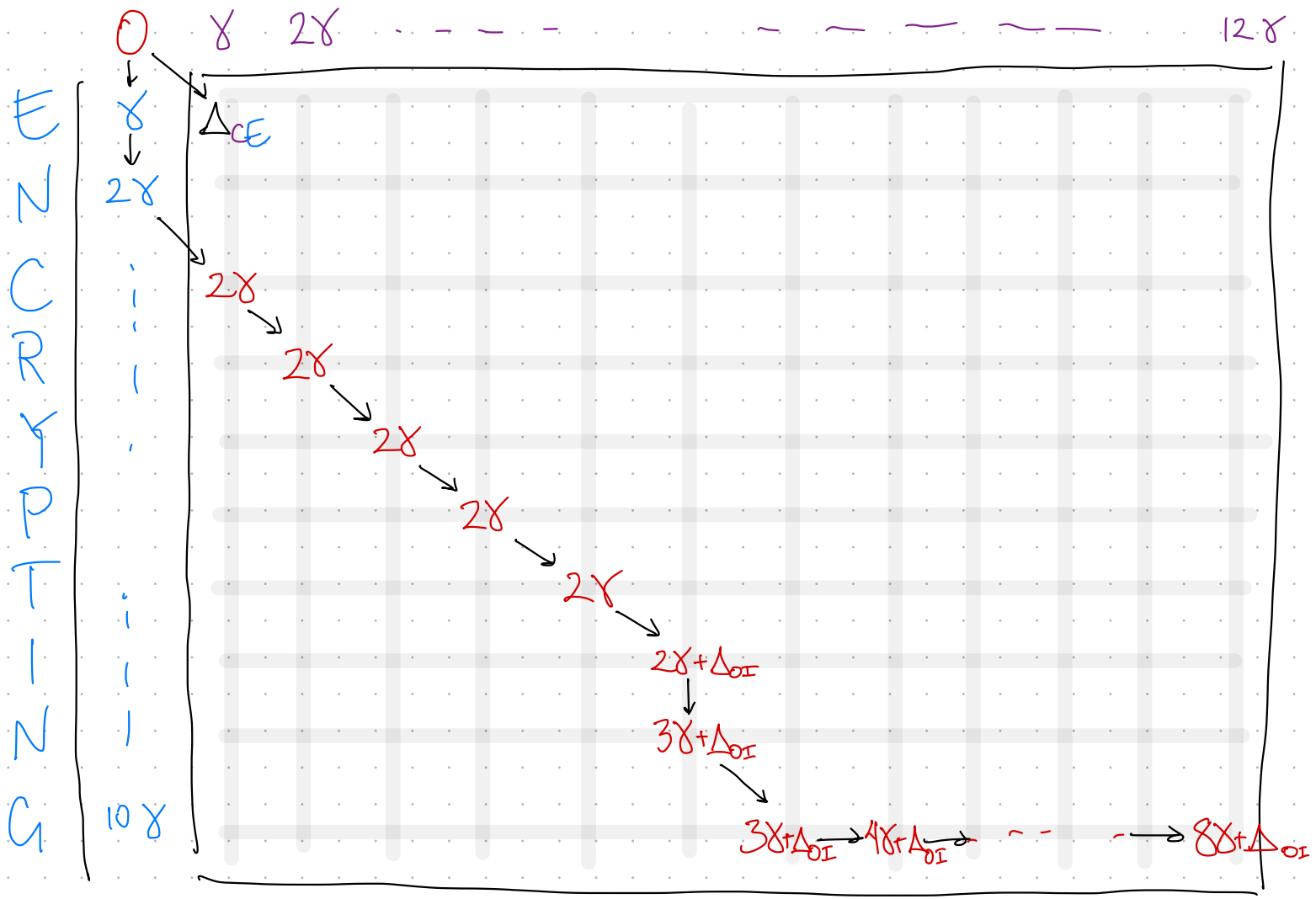
  For $j = 1 \longrightarrow m$

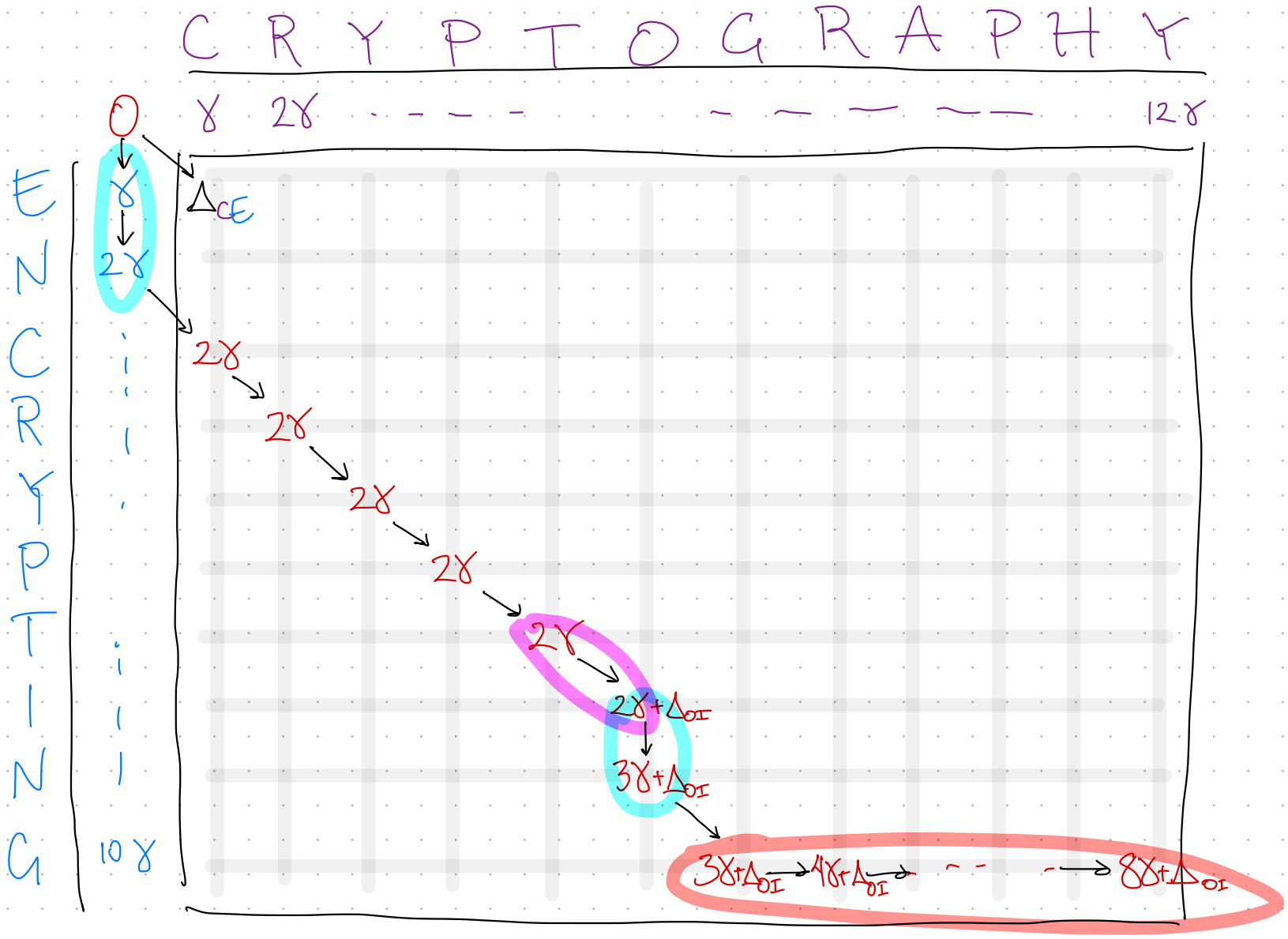
    $ED(i,j) = \min \{ ED(i-1, j-1) + \Delta_{s_i T_j} , ED(i-1, j) + \gamma , ED(i, j-1) + \gamma \}$

return $ED(n, m)$


insertions
Edit
+ + C R Y P T O + G R A P H Y
E N C R Y P T I N G
deletions

CRYPTOGRAPHY




CRYPTOGRAPHY
0
γ
2γ
12γ
ENCRYPTING
10γ
Δ_{CE}
2γ
2γ
2γ
2γ
2γ
$2\gamma+\Delta_{OI}$
$3\gamma+\Delta_{OI}$
$3\gamma+\Delta_{OI}$
$4\gamma+\Delta_{OI}$
$8\gamma+\Delta_{OI}$

Edit Distance Algorithm

$ED(0,0) = 0$

$ED(i,0) = i \cdot \gamma$

$ED(0,j) = j \cdot \gamma$

For $i = 1 \rightarrow n$

  For $j = 1 \rightarrow m$

    $ED(i,j) = \min \{ ED(i-1, j-1) + \Delta_{s_i T_j} , ED(i-1, j) + \gamma , ED(i, j-1) + \gamma \}$

return $ED(n, m)$

Running Time.

- $1 + n + m$ initialization
- n outer iterations
  m inner iterations
  $O(1)$ per iteration

Edit Distance Algorithm

$ED(0,0) = 0$

$ED(i,0) = i \cdot \gamma$

$ED(0,j) = j \cdot \gamma$

For $i = 1 \rightarrow n$

  For $j = 1 \rightarrow m$

    $ED(i,j) = \min \{ ED(i-1, j-1) + \Delta_{s_i T_j} , ED(i-1, j) + \gamma , ED(i, j-1) + \gamma \}$

return $ED(n, m)$

Running Time.

- $1 + n + m$ initialization
- n outer iterations
  m inner iterations
  $O(1)$ per iteration

Space

DP Table has $(m+1) \times (n+1)$ cells

$O(mn)$

Edit Distance ≡ Sequence Alignment

An alignment is a non-crossing matching.

ED Algorithm computed the distance

Can we compute the alignment?

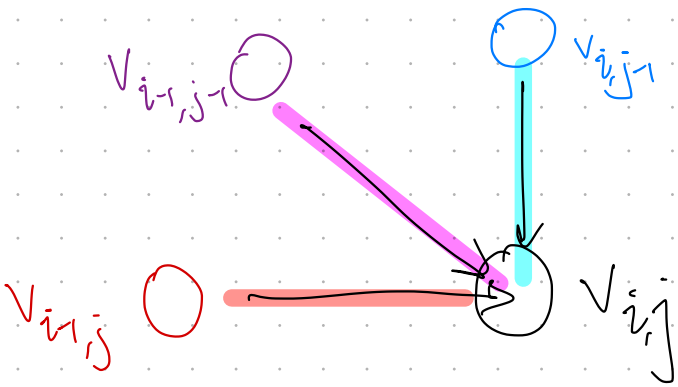
CRYPTOGRAPHY
ENCRYPTING
0
γ
2γ
12γ
10γ
Δ_{CE}
2γ
2γ
2γ
2γ
2γ
2γ
$2\gamma+\Delta_{OI}$
$3\gamma+\Delta_{OI}$
$3\gamma+\Delta_{OI}$
$4\gamma+\Delta_{OI}$
$8\gamma+\Delta_{OI}$
Idea.
Treat DP Table as a graph
entries ≡ vertices
transitions ≡ edges
Min ED alignment ≡ shortest path
diagonal edges ≡ alignment.

Theorem. Shortest path from $v_{00} \longrightarrow v_{ij}$ equals $ED(i,j)$.

Pf. By induction on $i+j$.

Base Case. Shortest path from $v_{00} \rightarrow v_{00} = 0$
$= ED(0,0)$

Inductive Step. Suppose shortest path to $v_{kl} = ED(k,l)$ for all $k+l < i+j$.



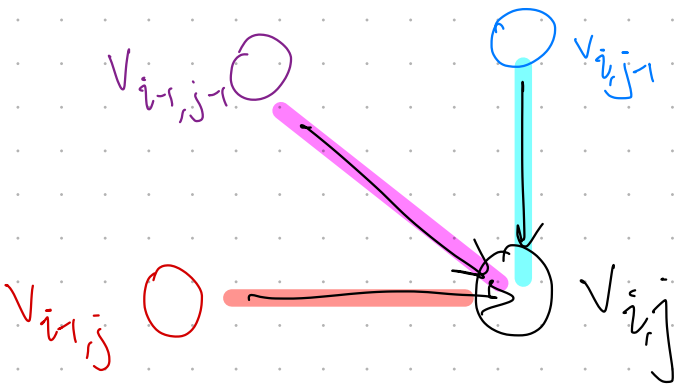
Consider SP to v_{ij}.
↳ only 3 in-edges.

Theorem. Shortest path from $v_{00} \longrightarrow v_{ij}$ equals $ED(i,j)$.

Pf. By induction on $i+j$.

Base Case. Shortest path from $v_{00} \to v_{00} = 0 = ED(0,0)$

Inductive Step. Suppose shortest path to $v_{kl} = ED(k,l)$ for all $k+l < i+j$.



Consider SP to v_{ij}.

↳ only 3 in-edges.

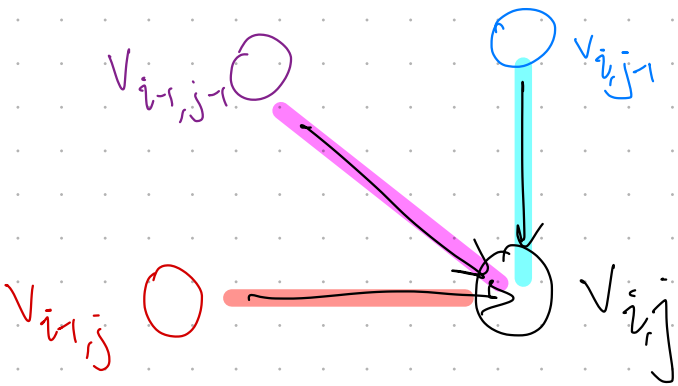
$$SP(v_{00} \to v_{ij}) = \min \begin{cases} SP(v_{00} \to v_{i-1,j-1}) + \Delta_{s_i t_j}, \\ SP(v_{00} \to v_{i-1,j}) + \gamma, \\ SP(v_{00} \to v_{i,j-1}) + \gamma \end{cases}$$

Theorem. Shortest path from $v_{00} \longrightarrow v_{ij}$ equals $ED(i,j)$.

Pf. By induction on $i+j$.

Base Case. Shortest path from $v_{00} \to v_{00} = 0 = ED(0,0)$

Inductive Step. Suppose shortest path to $v_{kl} = ED(k,l)$ for all $k+l < i+j$.



Consider SP to v_{ij}.

↳ only 3 in-edges.

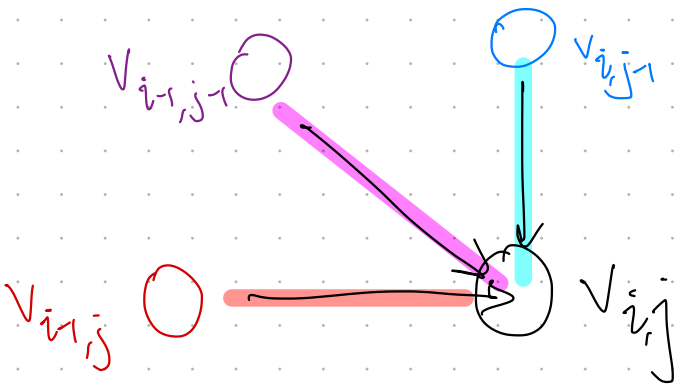
$$SP(v_{00} \to v_{ij}) = \min \left\{ \begin{array}{l} ED(i-1, j-1) + \Delta_{s_i T_j}, \\ ED(i-1, j) + \gamma, \\ ED(i, j-1) + \gamma \end{array} \right\}$$

Theorem. Shortest path from $v_{00} \longrightarrow v_{ij}$ equals $ED(i,j)$.

Pf. By induction on $i+j$.

Base Case. Shortest path from $v_{00} \to v_{00} = 0$
$= ED(0,0)$

Inductive Step. Suppose shortest path to $v_{kl} = ED(k,l)$ for all $k+l < i+j$.

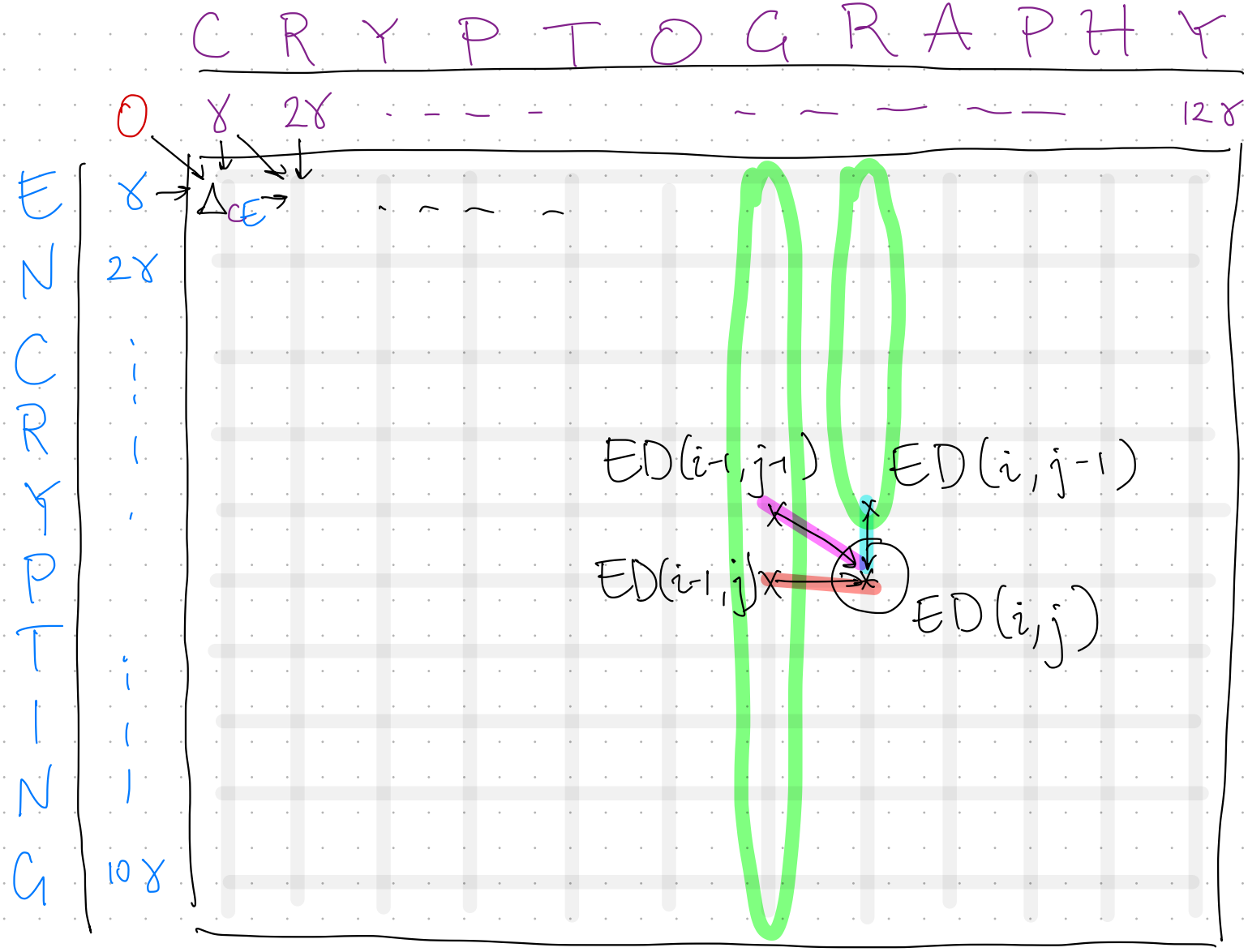


Consider SP to v_{ij}.
↳ only 3 in-edges.

$$SP(v_{00} \to v_{ij}) = ED[i,j]$$

∎

Reducing the Space Complexity?



Each entry requires 3 probes into prior entries.

↳ From prev. column from top to bottom.

Linear Space ED.

```
Prev(j) = j · γ                    // m-entry 1D arrays
Curr(j) = 0
For i = 1 → n
    Curr(0) = i · γ
    For j = 1 → m
        Curr(j) = min{ Prev(j-1) + δ_{s_i t_j} , Prev(j) + γ , Curr(j-1) + γ }
    Prev ← Curr.
return Curr(m).
```