Edit Distance

Outline

- Edit Distance
  - What is it?
  - Gap representation

- Recursive approach
  - Optimal substructure
  - Recursion
  - Turn it into dynamic programming

Outline

- Edit Distance
  - What is it?
  - Gap representation

- Recursive approach
  - Optimal substructure
  - Recursion
  - Turn it into dynamic programming

Learning objectives:
You are able to
- describe the gap representation of the difference between two strings.
- describe the intermediate solutions to edit distance as a grid like DAG
- design a dynamic programming algorithm for the edit distance problem

Comparing DNA Sequences

DNA Bases:	
Adenine	(A)
Thymine	(T)
Guanine	(G)
Cytosine	(C)

Comparing DNA Sequences

DNA Bases:

Adenine	(A)
Thymine	(T)
Guanine	(G)
Cytosine	(C)

Very important DNA sequence

TAGCCGAAATTATCCGG

TAGCCGAAATTATCCGG

Unknown sampled sequence

Comparing DNA Sequences

Very important DNA sequence

An easy question:
Are they the same?

TAGCCGAAATTATCCGG

TAGCCGAAATTATCCGG

Unknown sampled sequence

Comparing DNA Sequences

An easy question:
Are two DNA sequences the same?

Useless (?):
Errors in sampling
Mutations
Alignment

Known DNA

TAGCCGAAATTATCCGG

Sampled DNA

TAGCCCAA_TTAACCGGA

Comparing DNA Sequences

Edit Distance:
Are two DNA sequences similar?

Useful anwer:
Measurable distance

Known DNA

TAGCCGAAATTATCCGG

Sampled DNA

TAGCCCAA_TTAACCGGA

Edit Distance

Edit operations:
1. Insertion
2. Deletion
3. Substitution: change one letter to another

Edit distance of
1. TAGCCGAAATTATCCGG
2. AGCCCAATTAACCGGA

Edit Distance

Edit operations:

1. Insertion
2. Deletion
3. Substitution: change one letter to another

Edit distance of

1. TAGCCGAAAGTTAACCGGA
2. AGCCCAAGTTAACCGGA

Edit Distance

Edit operations:
1. Insertion
2. Deletion
3. Substitution: change one letter to another

Edit distance of
1. TAGCCGAAAGTTAACCGGA
2. AGCCCAAGTTAACCGGA

A bit hard to read…

Outline

- Edit Distance
  - What is it?
  - Gap representation

- Recursive approach
  - Optimal substructure
  - Recursion
  - Turn it into dynamic programming

Edit Distance – Gap Representation

```
T A G C C G A A A G T   A A C C G G
  A G C C C A A   G T T A A C C G G A
```

Edit operations:

1. Insertion
2. Deletion
3. Substitution: change one letter to another

Edit Distance – Gap Representation

```
T A G C C G A A A G T   A A C C G G
  A G C C C A A   G T T A A C C G G A
```

Edit distance is (at most) 5

Edit operations:

1. Insertion
2. Deletion
3. Substitution: change one letter to another

Edit Distance

Input:
Two strings S and T.

Question:
What is the minimum number of edits you need to do to turn string S into string T?

Edit operations:
1. Remove a letter
2. Insert a letter
3. Substitute a letter

Output:
The gap representation of the minimum number of edit operations needed to turn S into T.

Outline

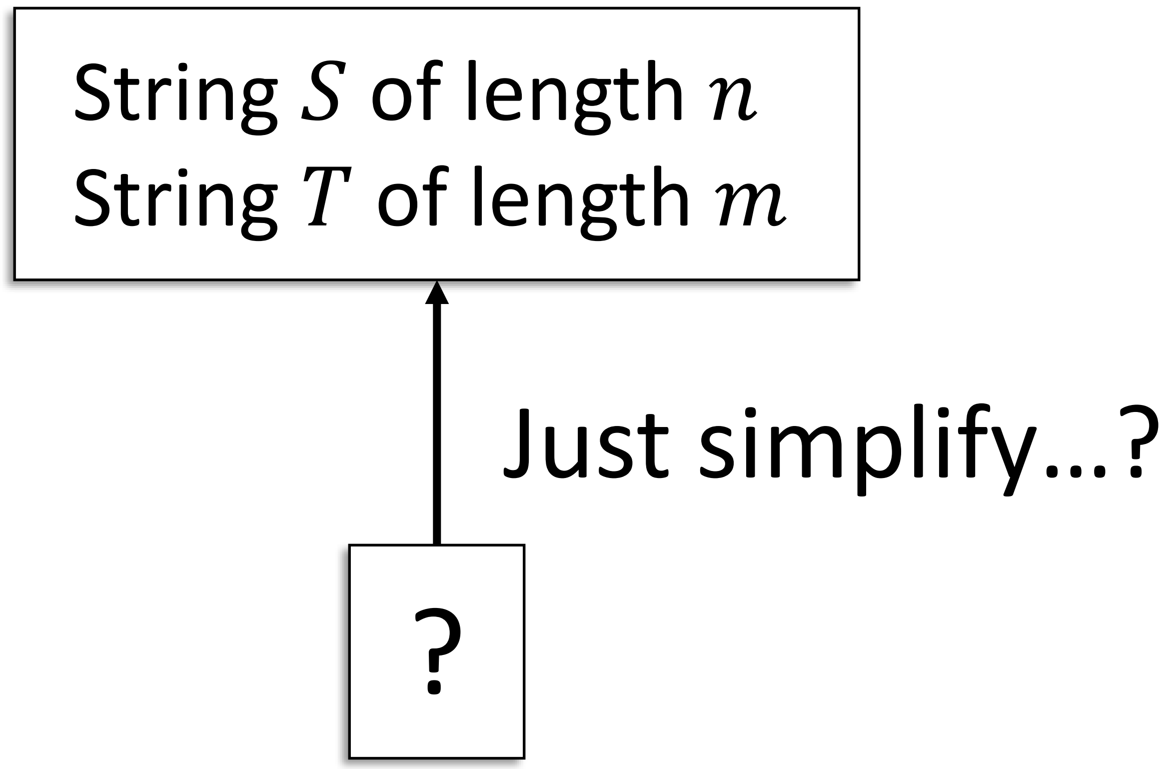
- Edit Distance
  - What is it?
  - Gap representation

- Recursive approach
  - Optimal substructure
  - Recursion
  - Turn it into dynamic programming

Recursive Approach

String S of length n
String T of length m

Recursive Approach



Recursive Approach

String S of length n
String T of length m

Just simplify…?

?

Pause and try to solve it yourselves.

Gap Representation – Optimal Substructure

T	A	G	C	C	G	A	A	A	G
	A	G	C	C	C	A	A		G

Lemma:
Suppose that you have an optimal gap representation.
Then, if you leave out the last column, the gap representation of the prefix is still optimal

Gap Representation – Optimal Substructure

The gray area is an optimal gap representation.

T	A	G	C	C	G	A	A	A	G
	A	G	C	C	C	A	A		G

Lemma:
Suppose that you have an optimal gap representation.
Then, if you leave out **the last column**, the gap representation of the prefix is still optimal

Gap Representation – Optimal Substructure

Lemma:
Suppose that you have an optimal gap representation.
Then, if you leave out the last column, the gap representation of the prefix is still optimal.

k edits in OPT

Gap Representation – Optimal Substructure

Case 1: last column is not an edit.

TAGCCGAAA	G
AGCCCAA	G

Lemma:
Suppose that you have an optimal gap representation.
Then, if you leave out the last column, the gap representation of the prefix is still optimal.

k edits in OPT

Gap Representation – Optimal Substructure

Lemma:
Suppose that you have an optimal gap representation.
Then, if you leave out the last column, the gap representation of the prefix is still optimal.

k edits in OPT

Case 1: last column is not an edit.

TAGCCGAAA	G
AGCCCAA	G

Prefix has k edits.

Gap Representation – Optimal Substructure

Case 1: last column is not an edit.

TAGCCGAAA	G
AGCCCAA	G

Lemma:
Suppose that you have an optimal gap representation.
Then, if you leave out the last column, the gap representation of the prefix is still optimal.

Prefix has k edits.

Suppose for a contradiction that prefix is not optimal.

k edits in OPT

Gap Representation – Optimal Substructure

Lemma:
Suppose that you have an optimal gap representation.
Then, if you leave out the last column, the gap representation of the prefix is still optimal.

k edits in OPT

Case 1: last column is not an edit.

TAGCCGAAA	G
AGCCCAA	G

Prefix has k edits.

Suppose for a contradiction that prefix is not optimal.

There must exist a gap representation P for the prefix with $< k$ edits.

Gap Representation – Optimal Substructure

Lemma:
Suppose that you have an optimal gap representation.
Then, if you leave out the last column, the gap representation of the prefix is still optimal.

k edits in OPT

Case 1: last column is not an edit.

TAGCCGAAA	G
AGCCCAA	G

Prefix has k edits.

Suppose for a contradiction that prefix is not optimal.

There must exist a gap representation P for the prefix with $< k$ edits.

Combine P with $\begin{smallmatrix} \text{G} \\ \text{G} \end{smallmatrix}$ to obtain a gap representation with $< k$ edits. A contradiction.

Gap Representation – Optimal Substructure

Lemma:
Suppose that you have an optimal gap representation.
Then, if you leave out the last column, the gap representation of the prefix is still optimal.

k edits in OPT

Case 1: last column is not an edit.

TAGCCGAAA	G
AGCCCAA	G

Prefix has k edits.

Suppose for a contradiction that prefix is not optimal.

There must exist a gap representation P for the prefix with $< k$ edits.

Combine P with $\begin{matrix}\text{G}\\\text{G}\end{matrix}$ to obtain a gap representation with $< k$ edits. A contradiction.

Gap Representation – Optimal Substructure

Lemma:
Suppose that you have an optimal gap representation.
Then, if you leave out the last column, the gap representation of the prefix is still optimal.

k edits in OPT

Case 2: last column is an edit.

TAGCCGAAA	T
AGCCCAA	G

Prefix has $k-1$ edits.

Suppose for a contradiction that prefix is not optimal.

There must exist a gap representation P for the prefix with $< k-1$ edits.

Combine P with $\begin{matrix}\text{T}\\\text{G}\end{matrix}$ to obtain a gap representation with $< k$ edits. A contradiction.

Outline

- Edit Distance
  - What is it?
  - Gap representation
- Recursive approach
  - Optimal substructure
  - Recursion
  - Turn it into dynamic programming

Simplify and Delegate

Lemma:
Prefixes of optimal gap representations are optimal.

Input :
Let $S = S[1 \dots n]$ and $T = T[1 \dots m]$

Simplify and Delegate

Lemma:
Prefixes of optimal gap representations are optimal.

Input :
Let $S = S[1 \dots n]$ and $T = T[1 \dots m]$

Simplify:
If we knew what the last operation is, we could solve the prefix and combine.

$S[1 \dots n]$	
$T[1 \dots m-1]$	$T[m]$

Simpler subproblem!

Simplify and Delegate

Lemma:
Prefixes of optimal gap representations are optimal.

Simplify:
If we knew what the last operation is, we could solve the prefix and combine.

All possibilities
All simpler

Insert

$S[1 \dots n]$	
$T[1 \dots m-1]$	$T[m]$

Delete

$S[1 \dots n-1]$	$S[n]$
$T[1 \dots m]$	

Substitute

$S[1 \dots n-1]$	$S[n]$
$T[1 \dots m-1]$	$T[m]$

Simplify and Delegate

Lemma:
Prefixes of optimal gap representations are optimal.

Simplify:
If we knew what the last operation is, we could solve the prefix and combine.

All possibilities
All simpler

Insert

$S[1 \ldots n]$	
$T[1 \ldots m-1]$	$T[m]$

Delete

$S[1 \ldots n-1]$	$S[n]$
$T[1 \ldots m]$	

Substitute

$S[1 \ldots n-1]$	$S[n]$
$T[1 \ldots m-1]$	$T[m]$

Simplify and Delegate

Lemma:
Prefixes of optimal gap representations are optimal.

Simplify:
If we knew what the last operation is, we could solve the prefix and combine.

All possibilities
All simpler

Insert

$S[1 \dots n]$	
$T[1 \dots m-1]$	$T[m]$

Delete

$S[1 \dots n-1]$	$S[n]$
$T[1 \dots m]$	

Substitute

$S[1 \dots n-1]$	$S[n]$
$T[1 \dots m-1]$	$T[m]$

"Do nothing" counts as a free substitution.

Recursive Algorithm

Edit($S[1 \dots n], T[1 \dots m]$):

if($n = 0$)

  return m

else if($m = 0$)

  return n

else

$$\text{return min} \begin{cases} \text{Edit}(S[1 \dots n], T[1 \dots m-1] + 1) \\ \text{Edit}(S[1 \dots n-1], T[1 \dots m] + 1) \\ \text{Edit}(S[1 \dots n-1], T[1 \dots m-1]) \end{cases}$$

Recursive Algorithm

Edit($S[1 \ldots n], T[1 \ldots m]$):
if($n = 0$)
  return m
else if($m = 0$)
  return n
else

$$\text{return min}\begin{cases}\text{Edit}(S[1 \ldots n], T[1 \ldots m-1] + 1) \\ \text{Edit}(S[1 \ldots n-1], T[1 \ldots m] + 1) \\ \text{Edit}(S[1 \ldots n-1], T[1 \ldots m-1])\end{cases}$$

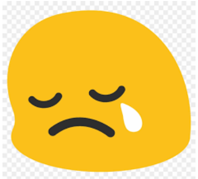
Runtime Recurrence

$$T(n,m) = \begin{cases} n & \text{if } m = 0 \\ m & \text{if } n = 0 \\ \min \begin{cases} \text{Edit}(S[1 \ldots n-1], T[1 \ldots m] + 1) \\ \text{Edit}(S[1 \ldots n], T[1 \ldots m-1] + 1) \\ \text{Edit}(S[1 \ldots n-1], T[1 \ldots m-1]) \end{cases} & \end{cases}$$

Runtime Recurrence

$$T(n,m) = \begin{cases} n & \text{if } m = 0 \\ m & \text{if } n = 0 \\ \min \begin{cases} \text{Edit}(S[1 \dots n-1], T[1 \dots m] + 1) \\ \text{Edit}(S[1 \dots n], T[1 \dots m-1] + 1) \\ \text{Edit}(S[1 \dots n-1], T[1 \dots m-1]) \end{cases} & \end{cases}$$

Three new calls per recursive call…
Probably something like: $T(n,m) \approx 3^{m+n}$



Outline

- Edit Distance
  - What is it?
  - Gap representation
- Recursive approach
  - Optimal substructure
  - Recursion
  - Turn it into dynamic programming

Dynamic Programming

Think for a bit:
Fix the inputs $S[1 \ldots n]$ and $T[1 \ldots m]$.

There are at most $n \cdot m$ distinct values for $\text{Edit}(\cdot,\ \cdot)$

Dynamic Programming

Think for a bit:
Fix the inputs $S[1 \dots n]$ and $T[1 \dots m]$.

There are at most $n \cdot m$ distinct values for Edit$(\cdot, \ \cdot)$

We can figure out Edit(n, m) in constant time if we know:
1. Edit$(n, m-1)$
2. Edit$(n-1, m)$
3. Edit$(n-1, m-1)$

Edit$(S[1 \dots n], T[1 \dots m])$:
if$(n = 0)$
  return m
else if$(m = 0)$
  return n
else
  return min

$$\begin{cases} \text{Edit}(S[1 \dots n], T[1 \dots m-1] + 1) \\ \text{Edit}(S[1 \dots n-1], T[1 \dots m] + 1) \\ \text{Edit}(S[1 \dots n-1], T[1 \dots m-1]) \end{cases}$$

Dynamic Programming

Think for a bit:
Fix the inputs $S[1 \ldots n]$ and $T[1 \ldots m]$.

There are at most $n \cdot m$ distinct values for Edit$(\cdot,\ \cdot)$

We can figure out Edit(n, m) in constant time if we know:

1. Edit$(n, m-1)$
2. Edit$(n-1, m)$
3. Edit$(n-1, m-1)$

Edit$(S[1 \ldots n], T[1 \ldots m])$:
if$(n = 0)$
 return m
else if$(m = 0)$
 return n
else
 return min

$$\begin{cases} \text{Edit}(S[1 \ldots n], T[1 \ldots m-1] + 1) \\ \text{Edit}(S[1 \ldots n-1], T[1 \ldots m] + 1) \\ \text{Edit}(S[1 \ldots n-1], T[1 \ldots m-1]) \end{cases}$$

Dynamic Programming

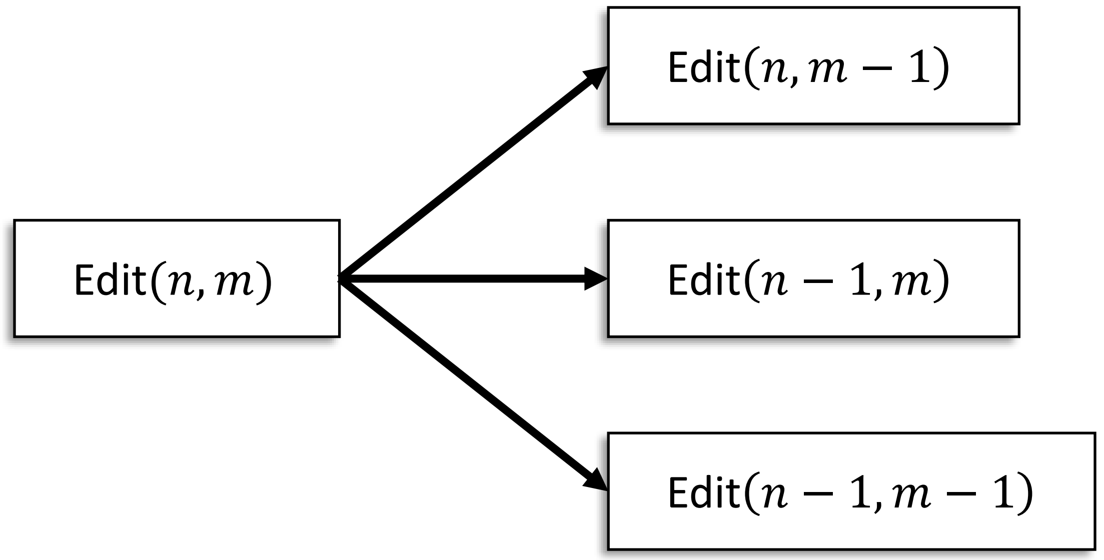
Think for a bit:
Fix the inputs $S[1 \ldots n]$ and $T[1 \ldots m]$.

There are at most $n \cdot m$ distinct values for Edit$(\cdot,\ \cdot)$

We can figure out Edit(n, m) in constant time if we know:

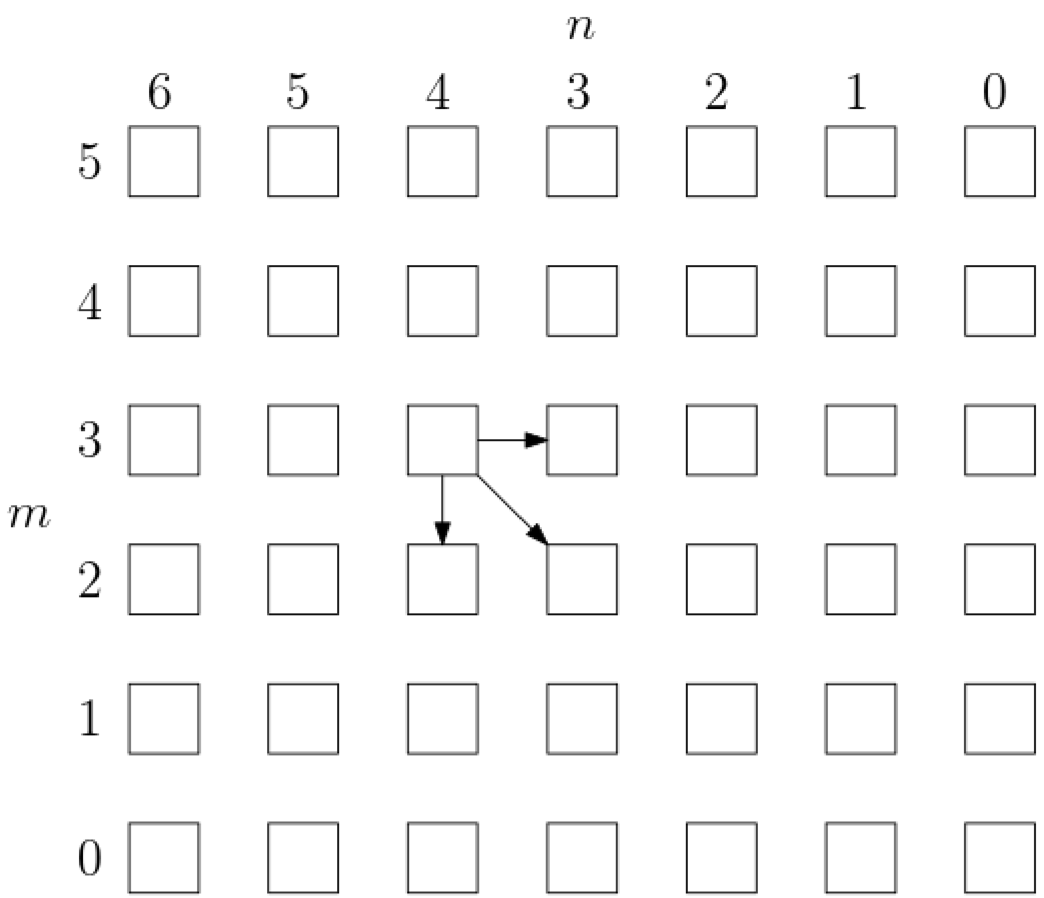
1. Edit$(n, m-1)$
2. Edit$(n-1, m)$
3. Edit$(n-1, m-1)$

DAG



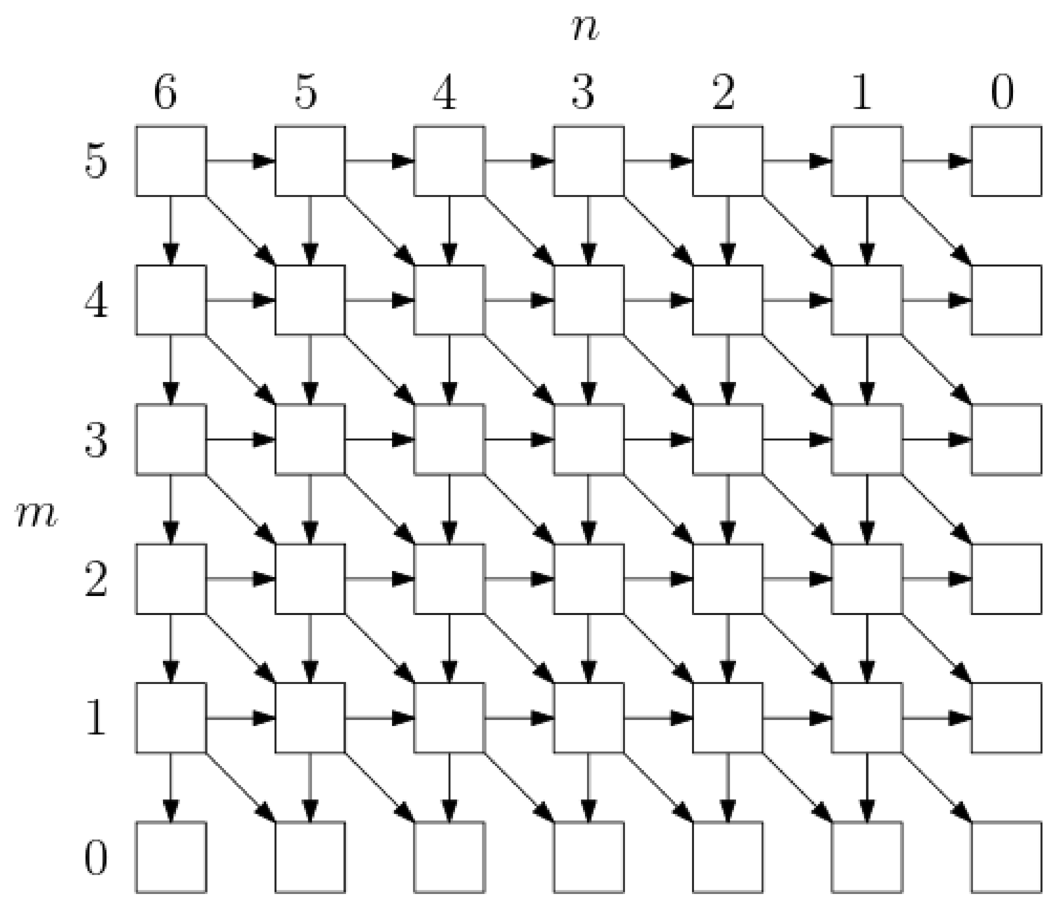
Dynamic Programming

Edit distance DAG:
A node depends on three other nodes.



Dynamic Programming

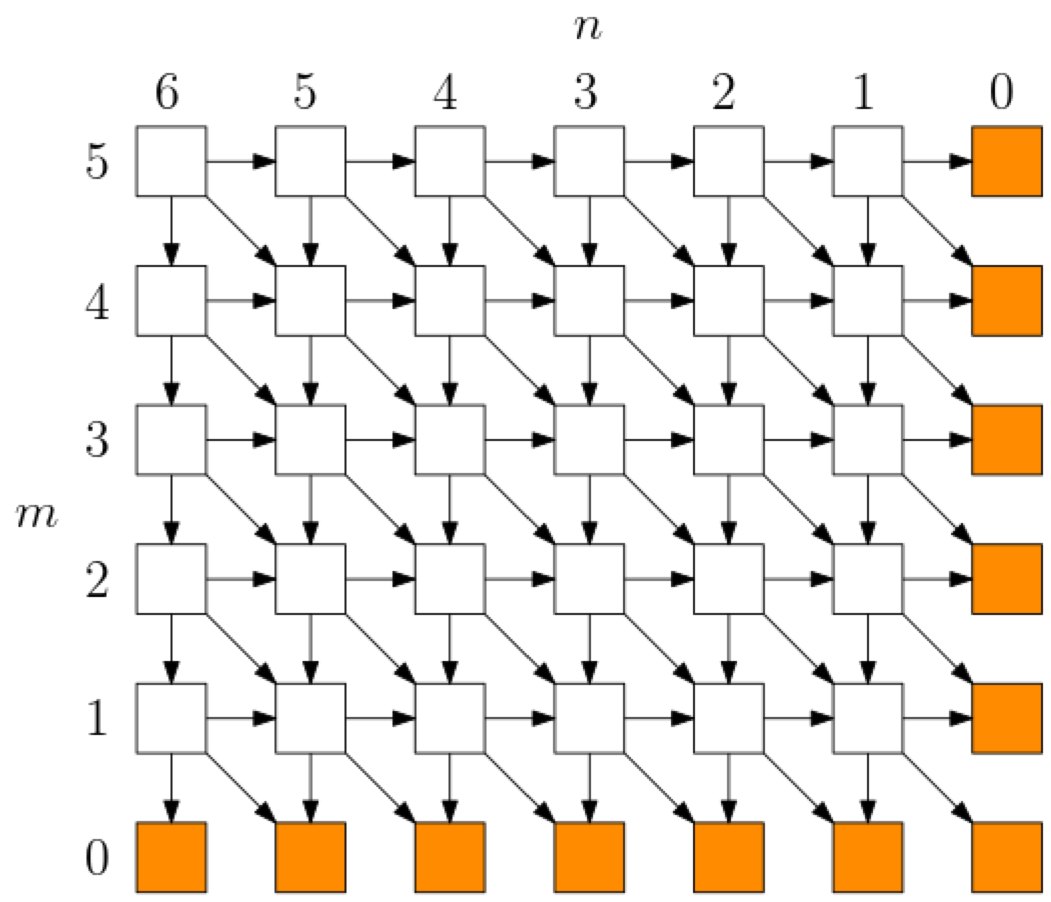
Edit distance DAG:
A node depends on three other nodes.



Initially, all nodes with a 0-coordinate are sinks.

Dynamic Programming

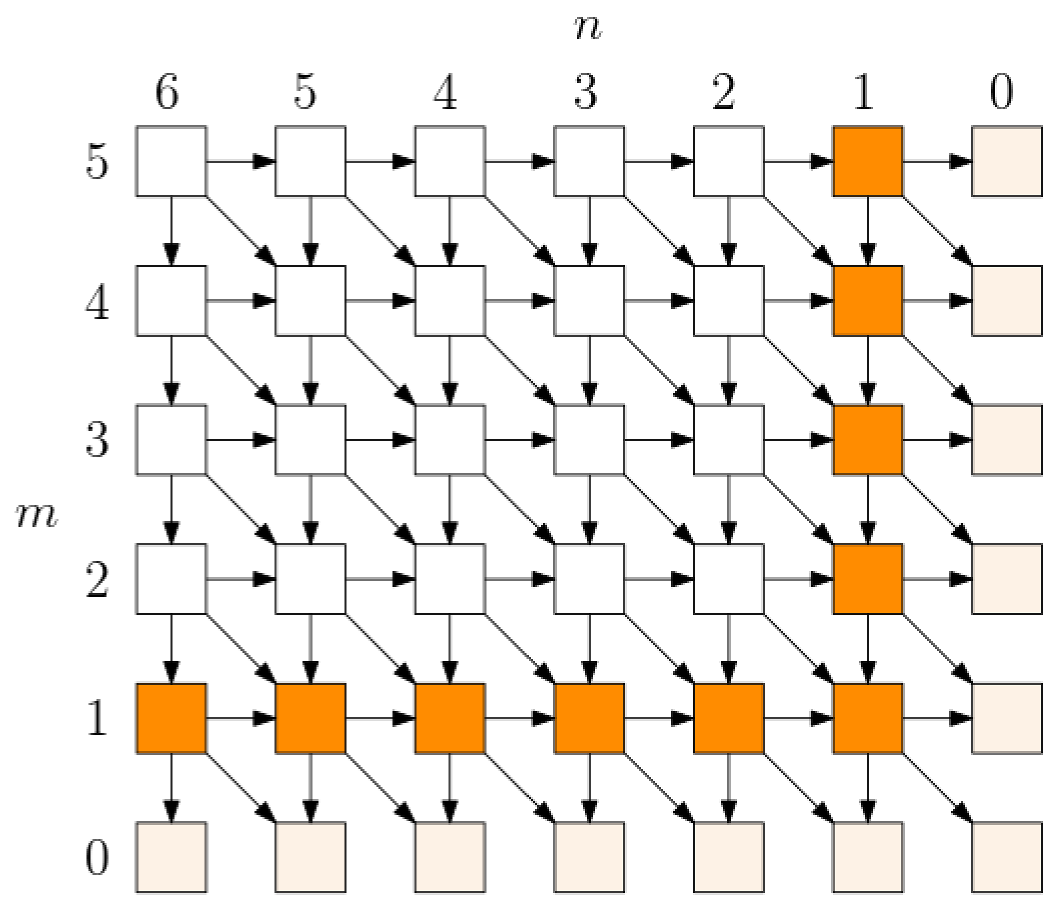
DP Algorithm:
Iteratively
solve sinks



Iteration 1

Dynamic Programming

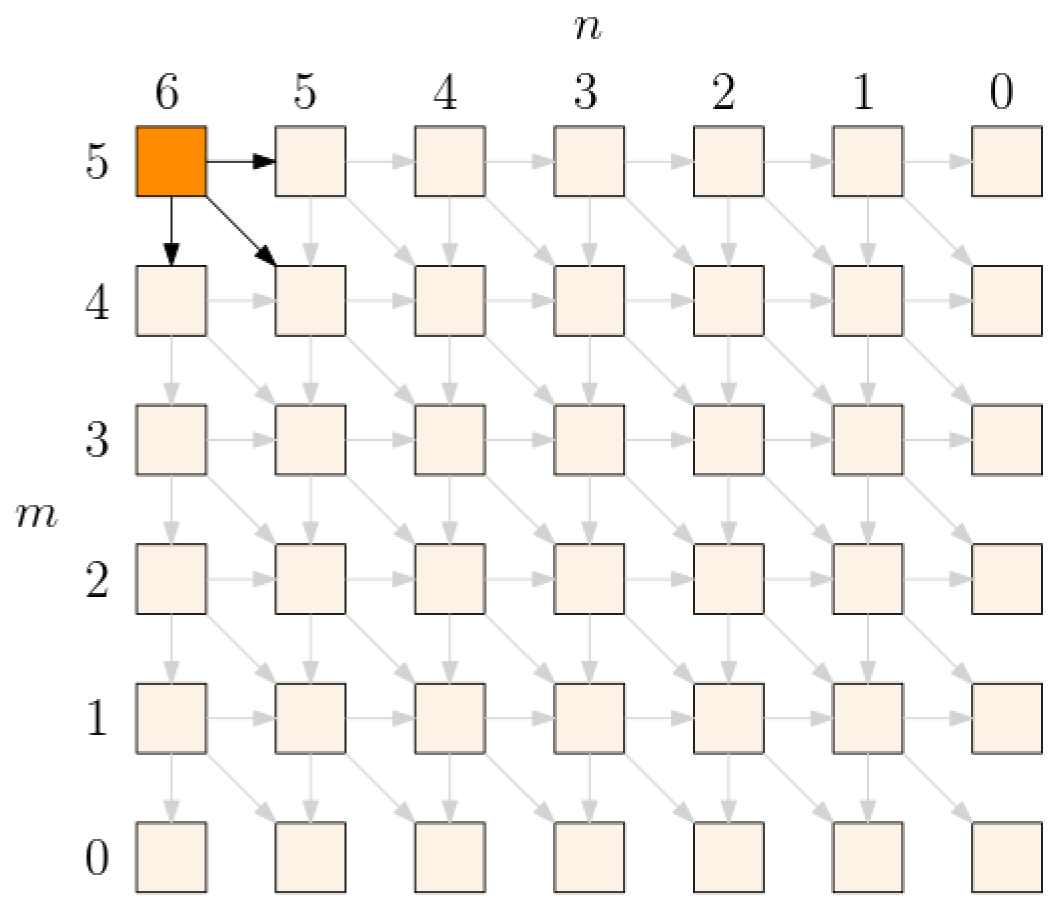
DP Algorithm:
Iteratively
solve sinks



Iteration 2

Dynamic Programming

DP Algorithm:
Iteratively
solve sinks



Last iteration

Edit Distance – Dynamic Programming

Correctness

Induction + optimal substructures

Runtime

Evaluating a sink takes constant time. The grid has $n \cdot m$ nodes.

The output

Edit Distance – Dynamic Programming

Correctness

Induction + optimal substructures

Runtime

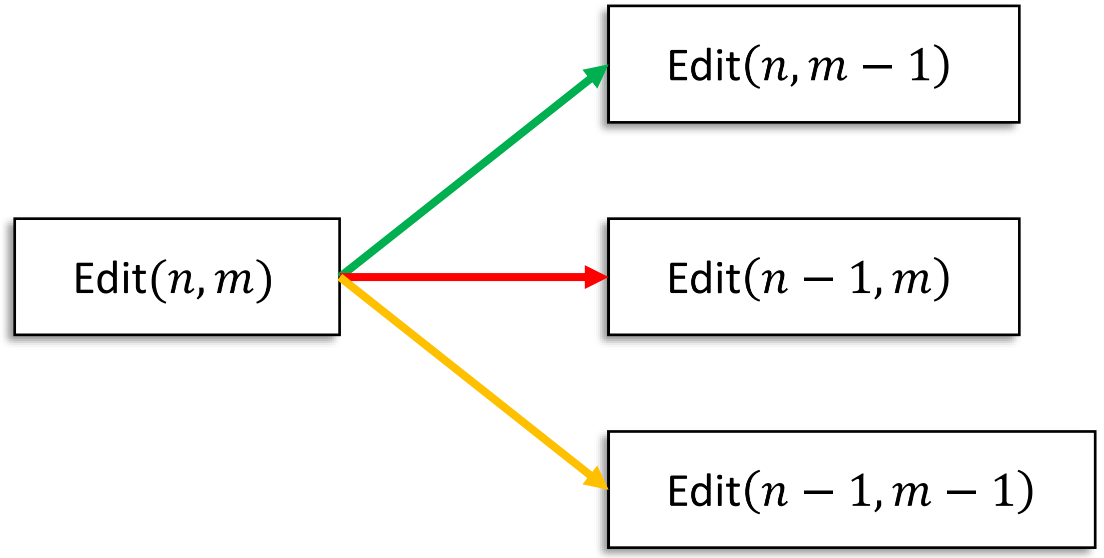
Evaluating a sink takes constant time. The grid has $n \cdot m$ nodes.

The output

Augment the grid with more information.

Dynamic Programming

DAG

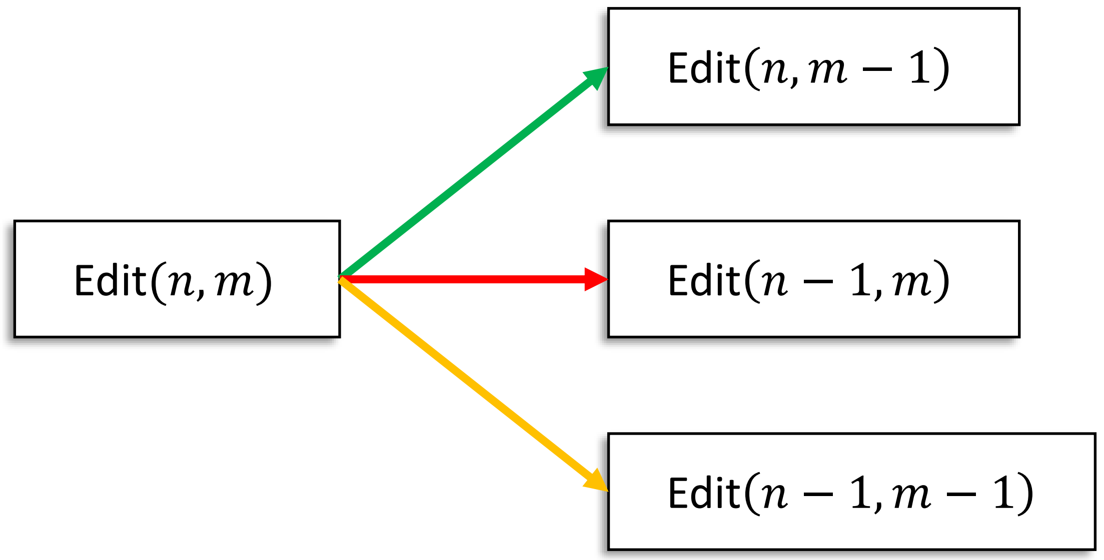


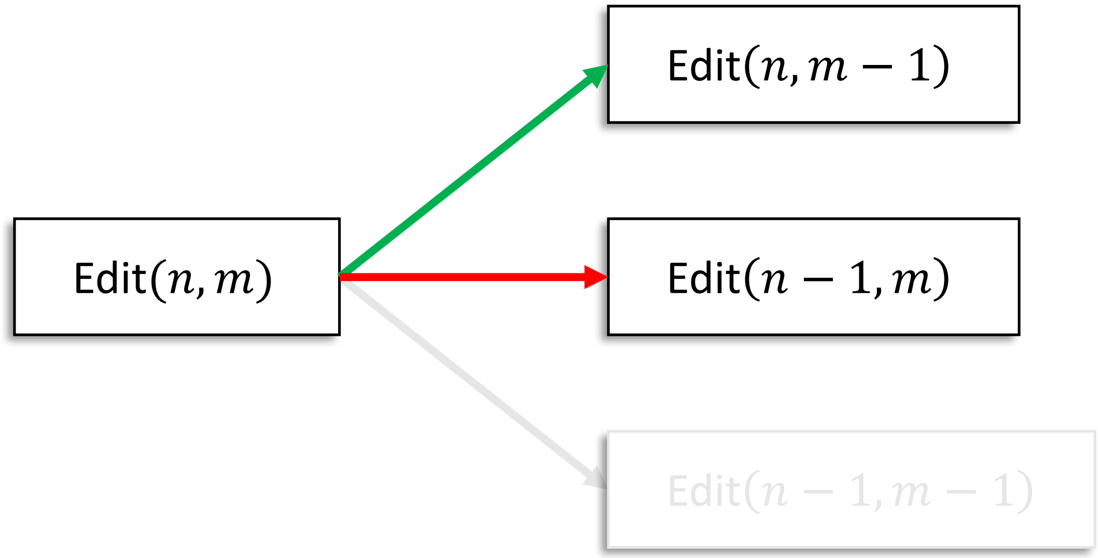
Edit operations:
1. Insertion
2. Deletion
3. Substitution

Dynamic Programming

Keep only the edge(s) that correspond to minimum cost operation

DAG





Edit operations:

1. Insertion
2. Deletion
3. Substitution

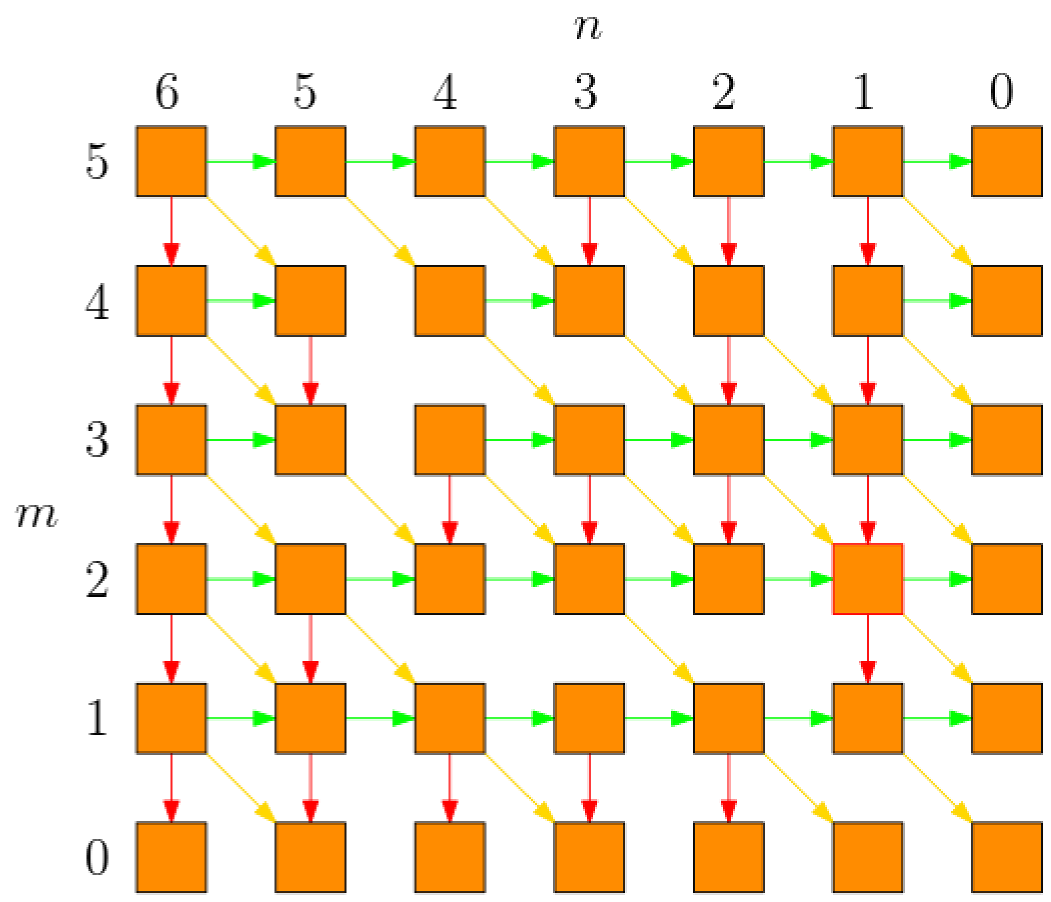
Edit($S[1 \dots n], T[1 \dots m]$):
if($n = 0$)
  return m
else if($m = 0$)
  return n
else
  return min $\begin{cases} \text{Edit}(S[1 \dots n], T[1 \dots m-1] + 1) \\ \text{Edit}(S[1 \dots n-1], T[1 \dots m] + 1) \\ \text{Edit}(S[1 \dots n-1], T[1 \dots m-1]) \end{cases}$

Dynamic Programming

On every node, write the cost.

On every edge, we have the operation

Follow cheapest arrows to reconstruct gap representation



In the book, in page 115, Chapter 3.7, you can find an example table for edit distance of ALGORITHM and ALTRUISTIC

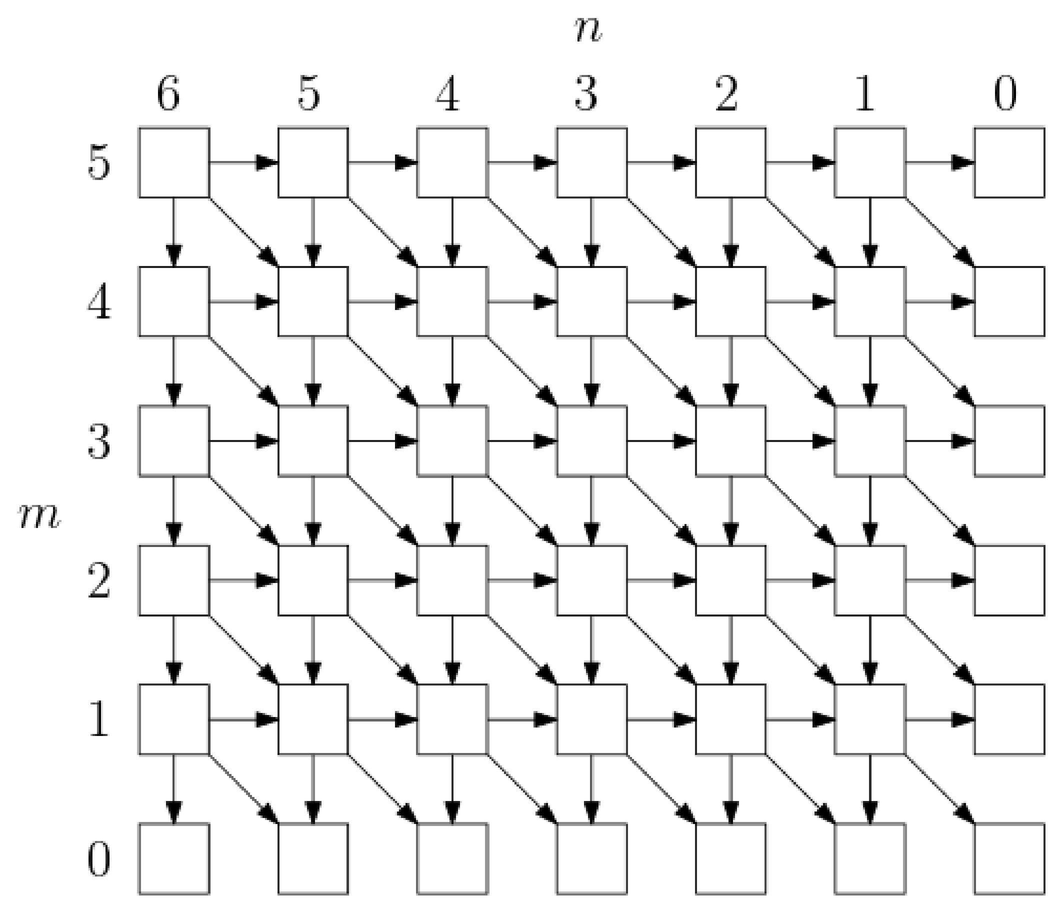
Wrap up

Edit distance

T A G C C G A A A G T _ A A C C G G

_ A G C C C A A _ G T T A A C C G G A

Lemma:
Prefix of an optimal gap representation is optimal.



Dynamic programming:
Iteratively evaluate the sinks of a memoization table.

Runtime: $O(n \cdot m)$