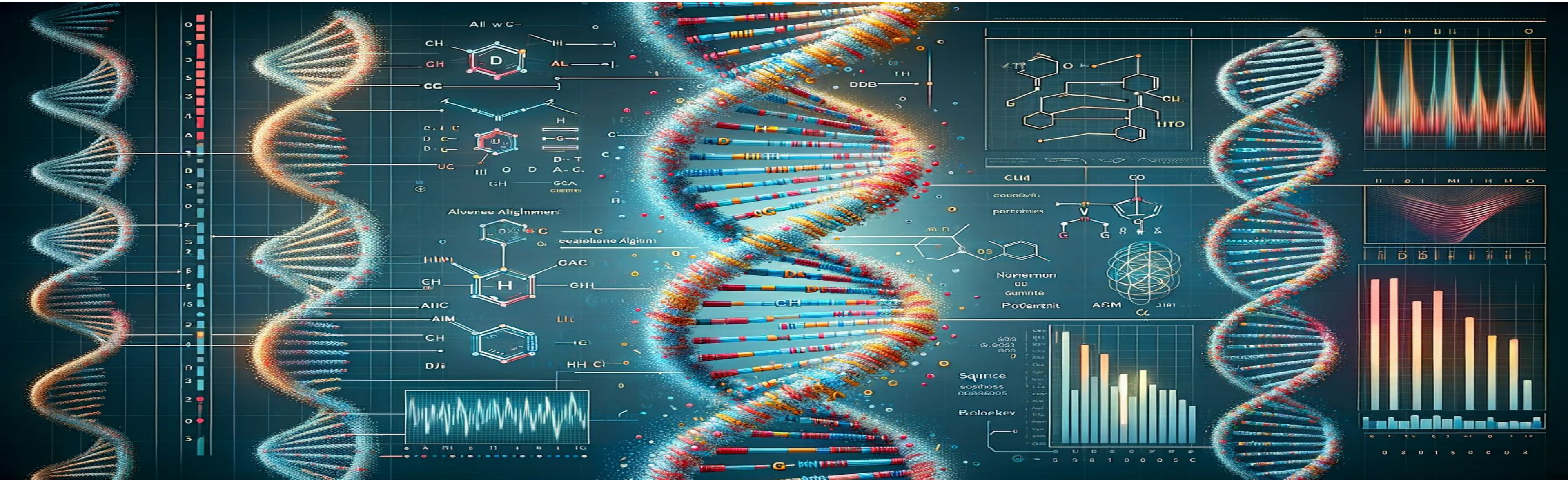
CSE7850/CX4803 Machine Learning in Computational Biology



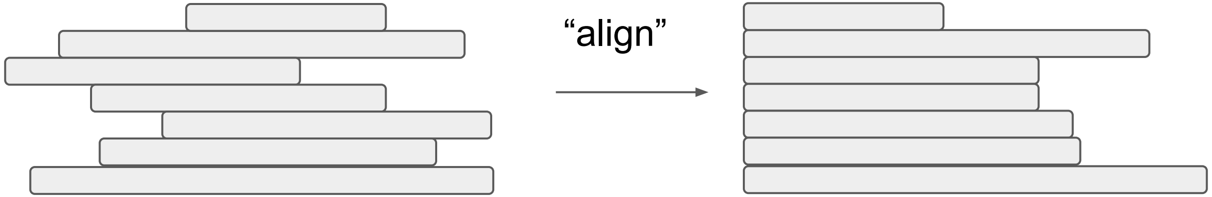
Lecture 3: Sequence Alignment

Yunan Luo

Acknowledgements: Some of the slides were adapted from lectures of Jian Peng and Mohammed El-Kebir

What is “alignment”

source



Alignment of protein/DNA sequences?

Given a set of sequences:

```
  ACTCGCAATATGC
    CTCCAATATGC
ATCGCTATATGC
       ACTGATTAC
 ACTCAATATGC
      TCGAATATGC
```

Sequence alignment is not only about this:

```
ACTCGCAATATGC
CTCCAATATGC
ATCGCTATATGC
ACTGATTAC
ACTCAATATGC
TCGAATATGC
```

Sequence alignment is more like this:

```
ACTCGCAATATGC
-CTC-CAATATGC
A-TCGCTATATGC
ACT-G-ATTA--C
ACT--CAATATGC
--TCG-AATATGC
```

Any patterns?

Alignment of protein/DNA sequences?

Given a set of sequences:

```
 ACTCGCAATATGC
   CTCCAATATGC
ATCGCTATATGC
     ACTGATTAC
 ACTCAATATGC
    TCGAATATGC
```

Sequence alignment is not only about this:

```
ACTCGCAATATGC
CTCCAATATGC
ATCGCTATATGC
ACTGATTAC
ACTCAATATGC
TCGAATATGC
```

Sequence alignment is more like this:

```
ACTCGCAATATGC
-CTC-CAATATGC
A-TCGCTATATGC
ACT-G-ATTA--C
ACT--CAATATGC
--TCG-AATATGC
```

Any patterns?

A real alignment

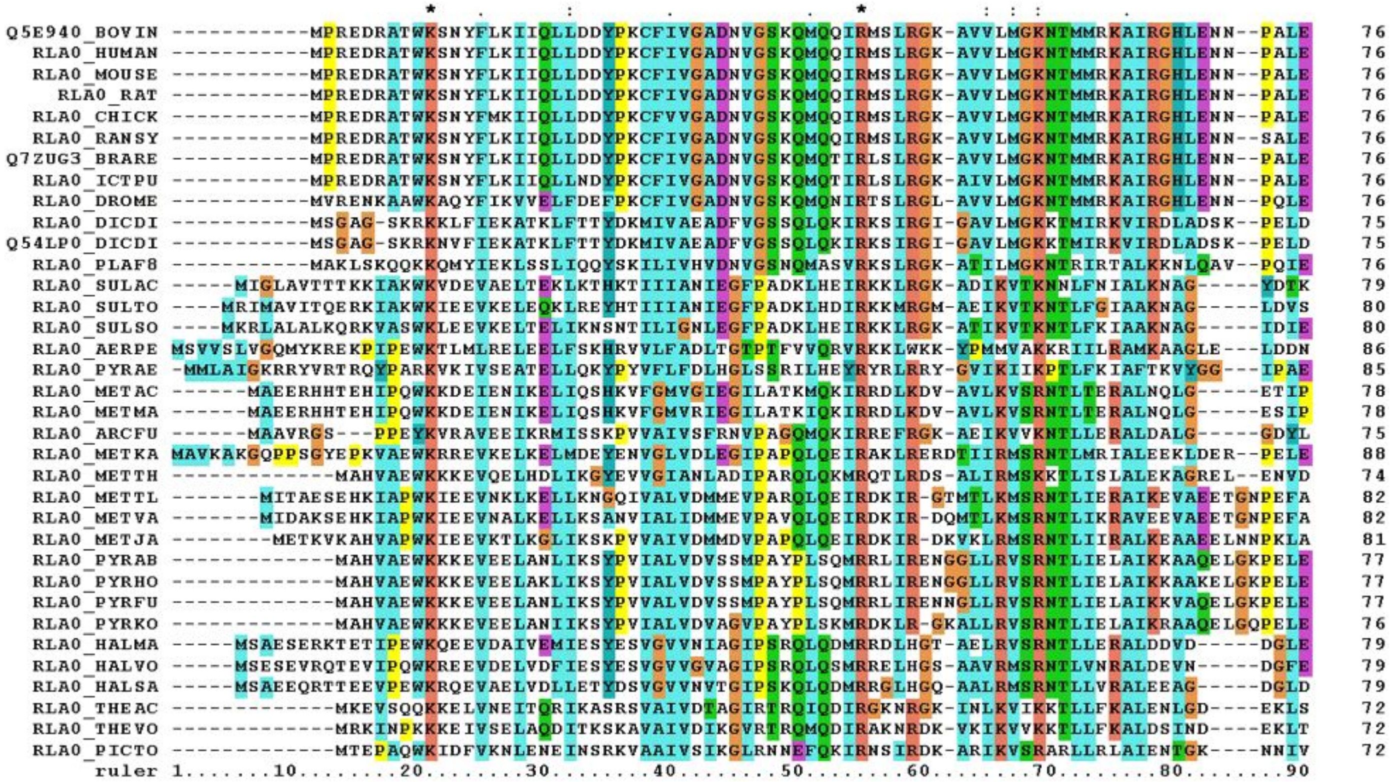
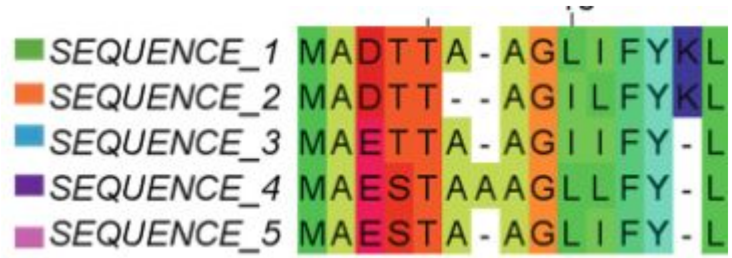
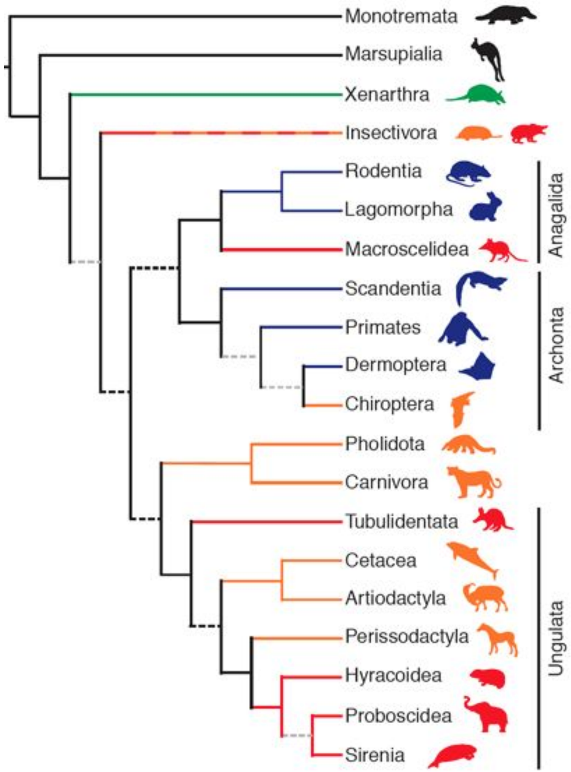


Image from Wiki

Why alignments?

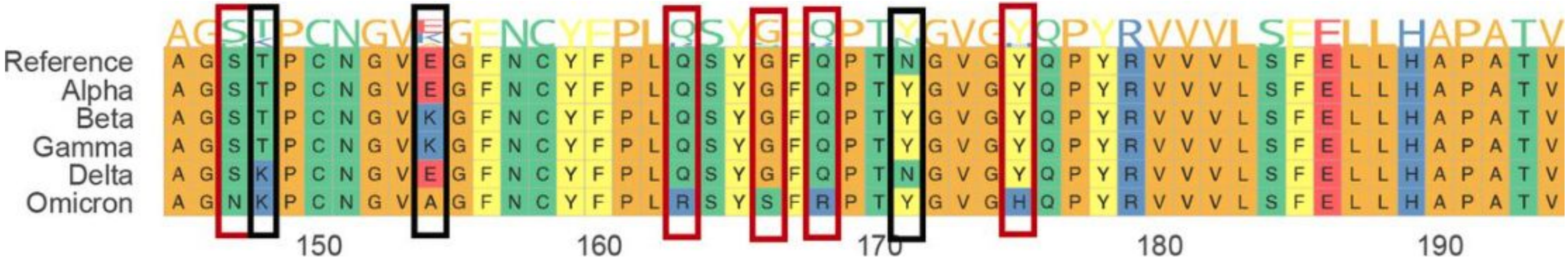
Example #1: reconstructing phylogenetic tree of species





Why alignments?

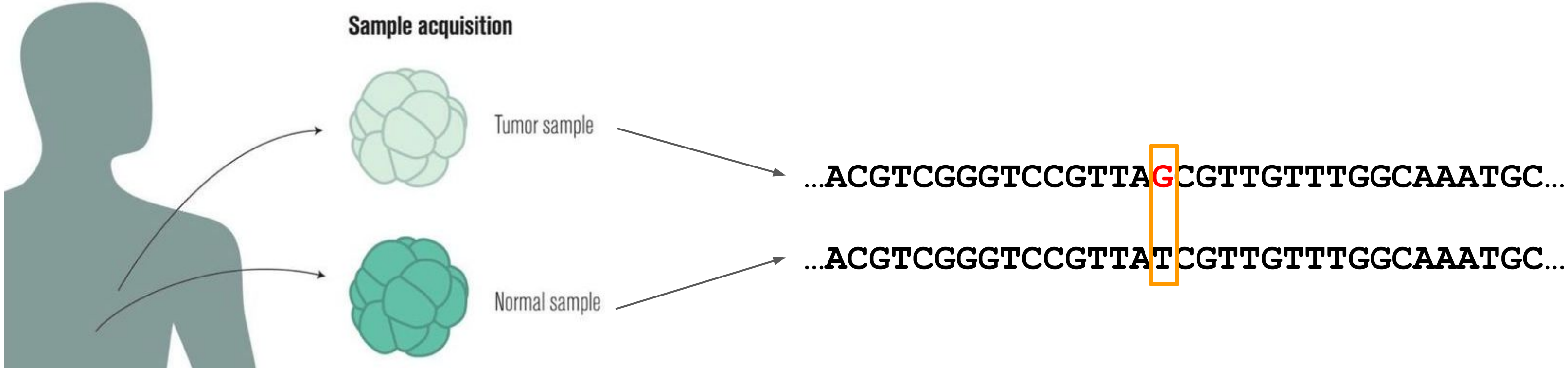
Example #2: comparing different COVID-19 variants



https://www.biorxiv.org/content/10.1101/2021.12.08.471688v1.full

Why alignments?

Example #3: identifying disease variants



“Goodness” of alignments

Given two sequences:

```
ATTTTCCC
ATTTACGC
```

There are many possible alignments: ***which one is better?***

```
ATTTTCCC
ATTTACGC
```

```
ATTT-TCCC
ATTTA-CGC
```

```
ATTTTCCC--------
--------ATTTACGC
```

Edit distance: the total number of substitutions, insertions and deletions needed to transform one sequence to another

“Goodness” of alignments

Given two sequences:

```
ATTTTCCC
ATTTACGC
```

There are many possible alignments: ***which one is better?***

```
ATTTTCCC
ATTTACGC
```

Edit distance=2

```
ATTT-TCCC
ATTTA-CGC
```

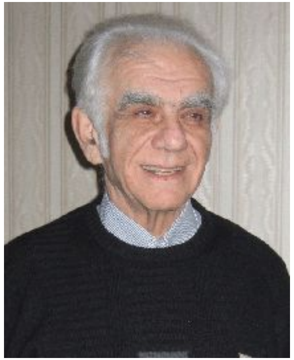
Edit distance=3

```
ATTTTCCC--------
--------ATTTACGC
```

Edit distance=16

Edit distance: the total number of substitutions, insertions and deletions needed to transform one sequence to another

Edit distance (Levenshtein, 1966)



Vladimir I. Levenshtein
(1935 - 2017)

Elementary operations: insertion, deletions and substitutions of single characters

Edit Distance problem: Given strings $\boldsymbol{v} \in \Sigma^m$ and $\boldsymbol{w} \in \Sigma^m$, compute the minimum number $d(\boldsymbol{v}, \boldsymbol{w})$ of elementary operations to transform $\boldsymbol{v}$ into $\boldsymbol{w}$.

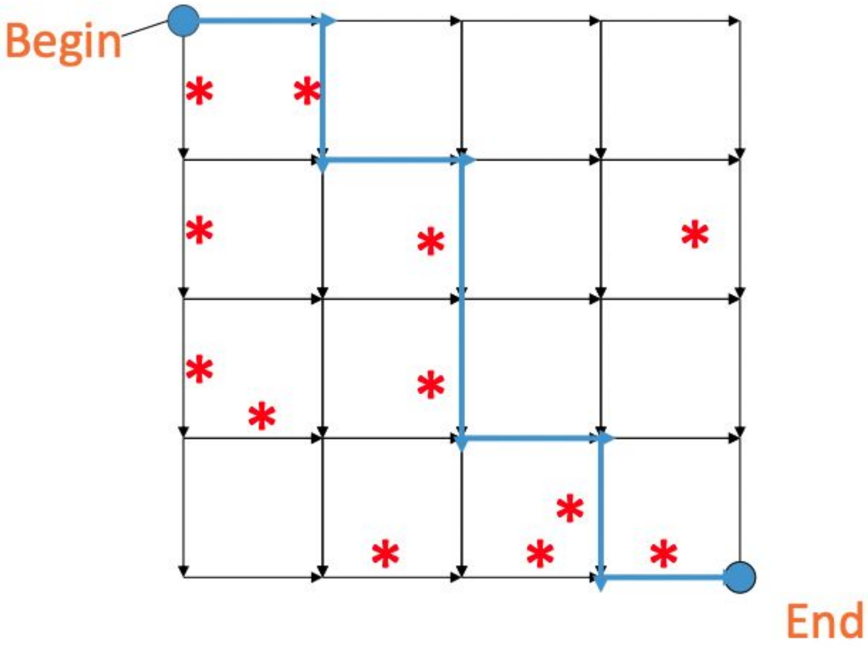
$$d(\mathbf{cat}, \mathbf{car}) = 1 \quad d(\mathbf{cat}, \mathbf{ate}) = 2 \quad d(\mathbf{cat}, \mathbf{are}) = 3$$

How to compute the edit distance?

$d($ **`SYKVKLITPDGPIEFDCPDD`**,
`AYKVTLVTPEGKQELECPDD` $) = ?$

Manhattan Tourist Problem

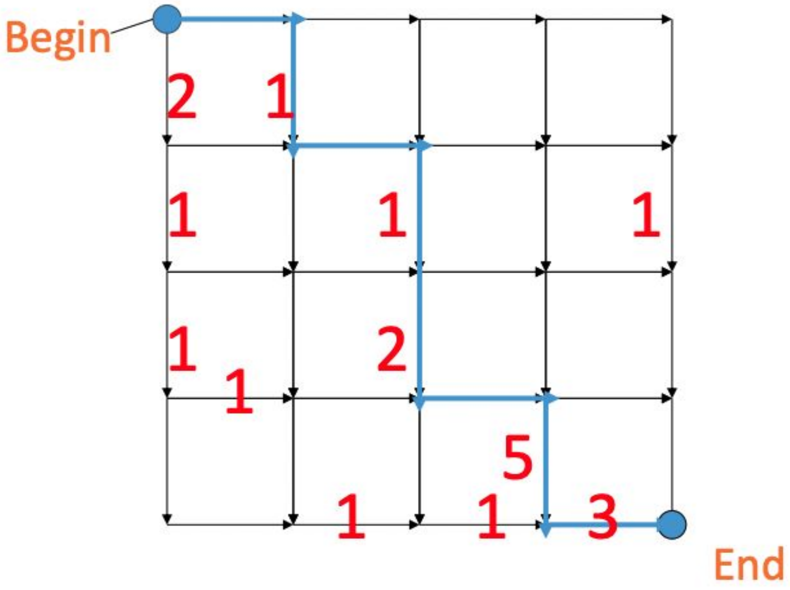
A tourist in Manhattan wants to visit the maximum number of attractions (*) by traveling on a path (only eastward and southward) from start to end



Manhattan Tourist Problem

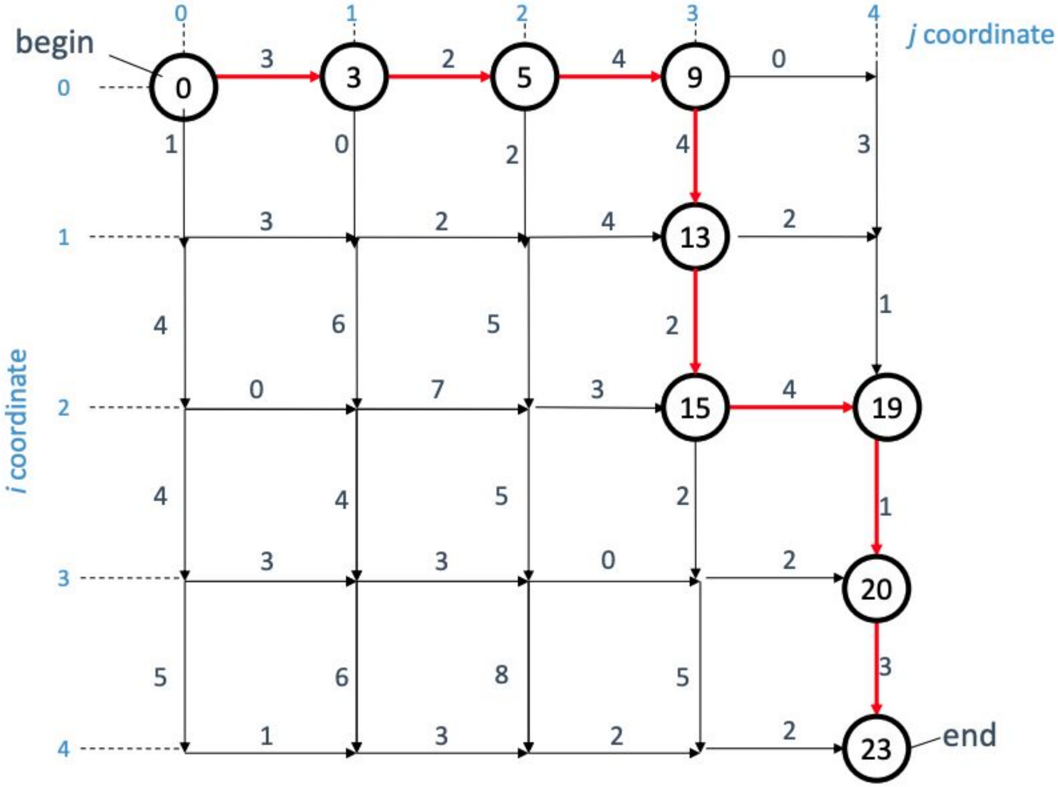
A tourist in Manhattan wants to visit the maximum number of attractions (*) by traveling on a path (only eastward and southward) from start to end

May be more than 1 attraction on a street. Add weights!



Manhattan Tourist Problem

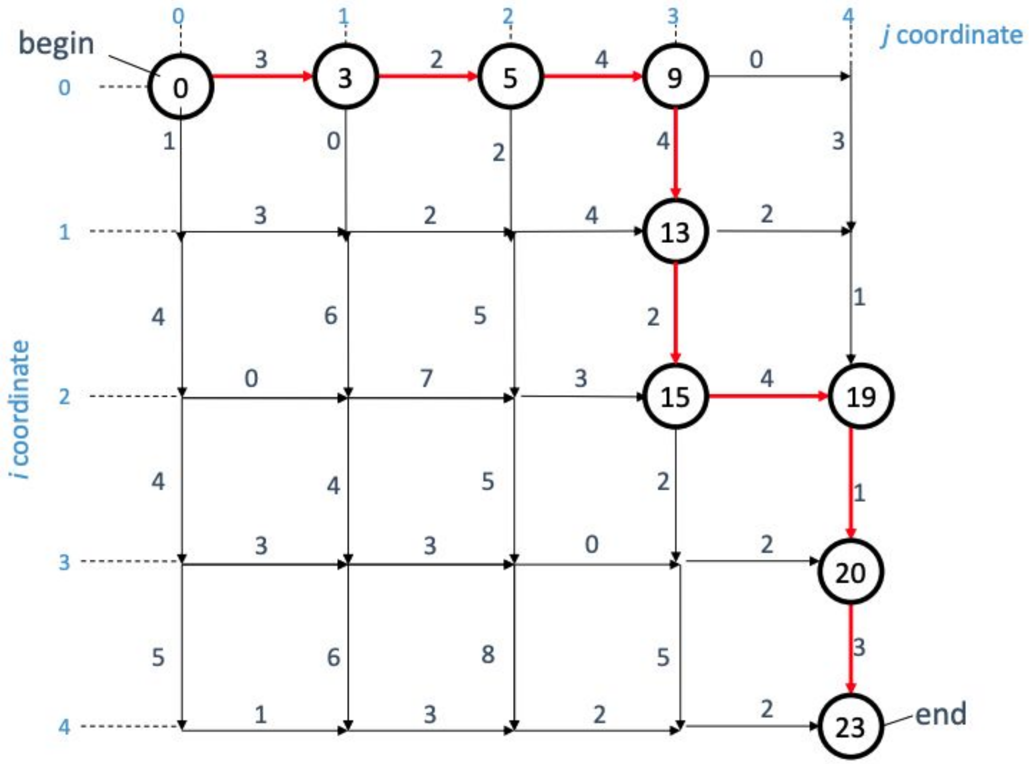
Manhattan Tourist Problem (MTP):
Given a weighted, directed grid graph *G* with two vertices “begin” and “end”, find the maximum weight path in *G* from “begin” to “end”.



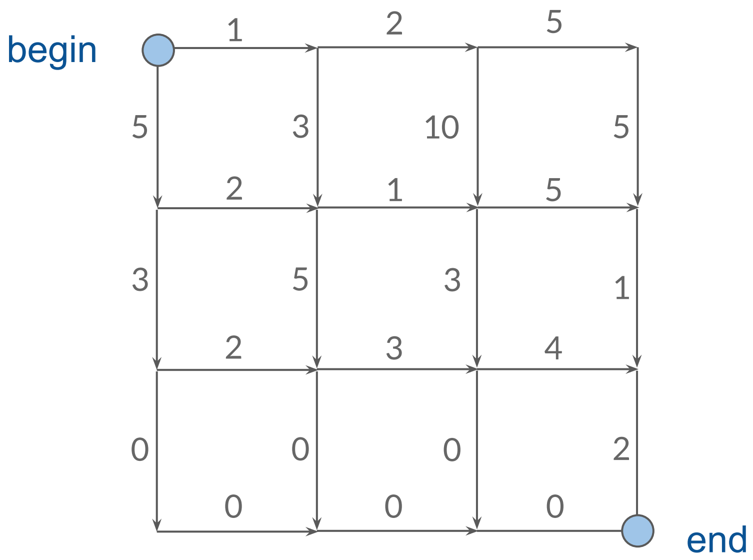
Solution - Exhaustive algorithm

Check all paths

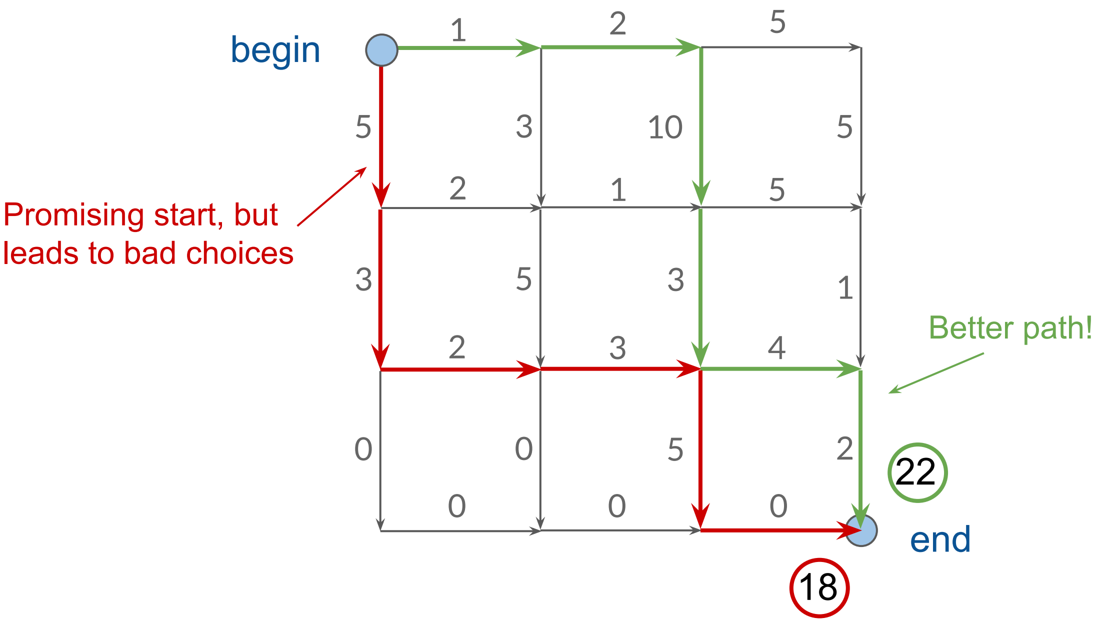
Question:
How many paths?



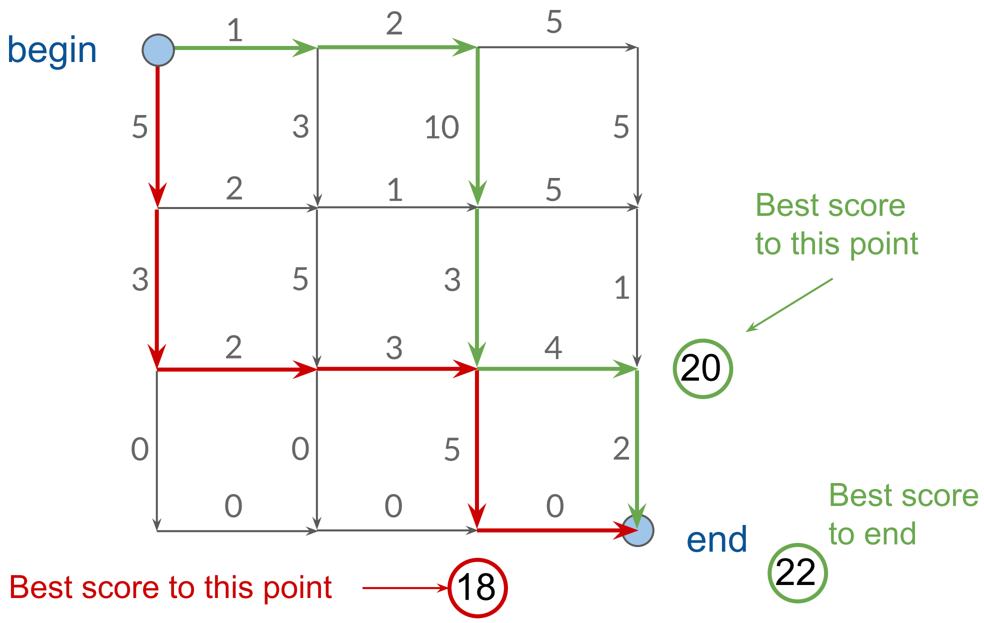
Solution - Greedy Algorithm



Solution - Greedy Algorithm



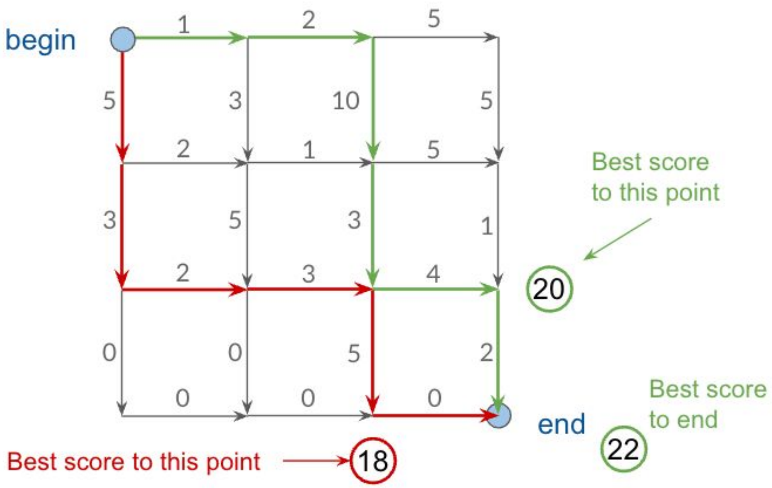
Optimal substructure



Optimal substructure

$s[i, j]$ is the best score for path to coordinate (i, j)

Question: What is the recurrence?

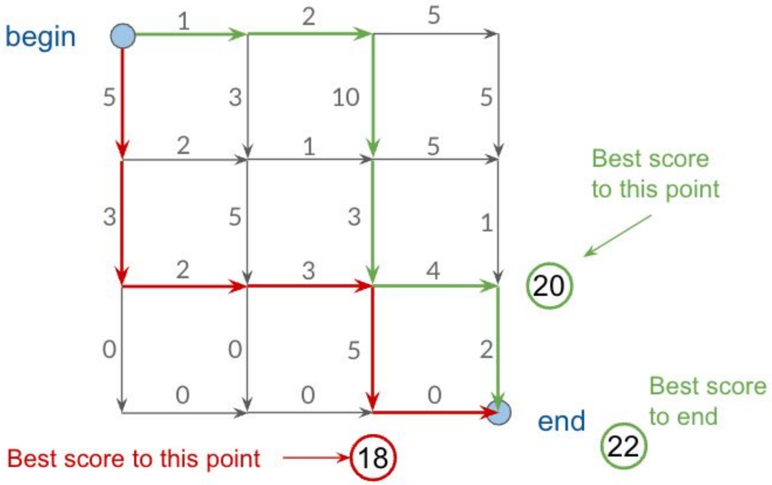


- $w[(i - 1, j), (i, j)]$ is weight of street between $(i - 1, j)$ and (i, j)
- $w[(i, j - 1), (i, j)]$ is weight of street between $(i, j - 1)$ and (i, j)

Optimal substructure

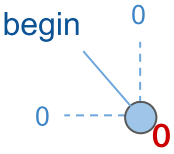
$s[i, j]$ is the best score for path to coordinate (i, j)

$$s[i,j] = \max \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ s[i-1,j] + w[(i-1,j),(i,j)] & \text{if } i > 0, \\ s[i,j-1] + w[(i,j-1),(i,j)] & \text{if } j > 0. \end{cases}$$



- $w[(i - 1, j), (i, j)]$ is weight of street between $(i - 1, j)$ and (i, j)
- $w[(i, j - 1), (i, j)]$ is weight of street between $(i, j - 1)$ and (i, j)

MTP -- Solving recurrence using **dynamic programming**

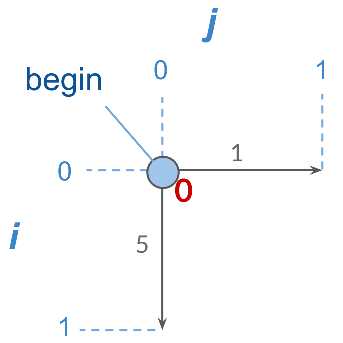


s[*i*, *j*] is the best score for path to coordinate (*i*, *j*)

$$s[i,j] = \max \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ s[i-1,j] + w[(i-1,j),(i,j)] & \text{if } i > 0, \\ s[i,j-1] + w[(i,j-1),(i,j)] & \text{if } j > 0. \end{cases}$$

- *w*[(*i* - 1, *j*), (*i*, *j*)] is weight of street between (*i* − 1, *j*) and (*i*, *j*)
- *w*[(*i*, *j* - 1), (*i*, *j*)] is weight of street between (*i*, *j* − 1) and (*i*, *j*)

MTP -- Solving recurrence using **dynamic programming**

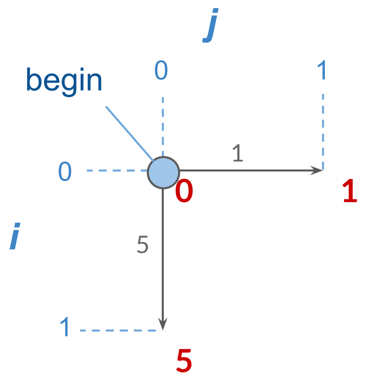


s[*i*, *j*] is the best score for path to coordinate (*i*, *j*)

$$s[i,j] = \max \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ s[i-1,j] + w[(i-1,j),(i,j)] & \text{if } i > 0, \\ s[i,j-1] + w[(i,j-1),(i,j)] & \text{if } j > 0. \end{cases}$$

- *w*[(*i* - 1, *j*), (*i*, *j*)] is weight of street between (*i* − 1, *j*) and (*i*, *j*)
- *w*[(*i*, *j* - 1), (*i*, *j*)] is weight of street between (*i*, *j* − 1) and (*i*, *j*)

MTP -- Solving recurrence using **dynamic programming**

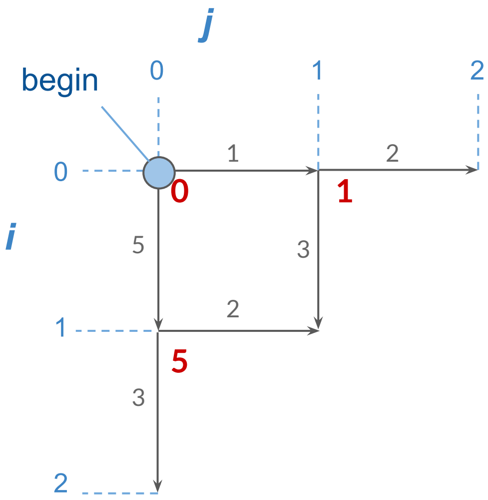


s[*i*, *j*] is the best score for path to coordinate (*i*, *j*)

$$s[i,j] = \max \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ s[i-1,j] + w[(i-1,j),(i,j)] & \text{if } i > 0, \\ s[i,j-1] + w[(i,j-1),(i,j)] & \text{if } j > 0. \end{cases}$$

- *w*[(*i* - 1, *j*), (*i*, *j*)] is weight of street between (*i* − 1, *j*) and (*i*, *j*)
- *w*[(*i*, *j* - 1), (*i*, *j*)] is weight of street between (*i*, *j* − 1) and (*i*, *j*)

MTP -- Solving recurrence using **dynamic programming**

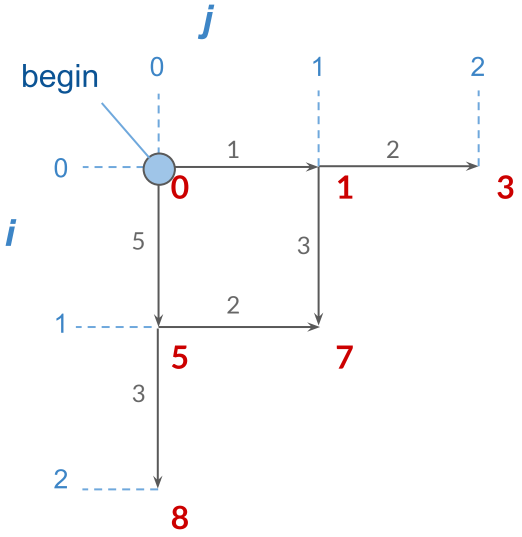


s[*i*, *j*] is the best score for path to coordinate (*i*, *j*)

$$s[i,j] = \max \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ s[i-1,j] + w[(i-1,j),(i,j)] & \text{if } i > 0, \\ s[i,j-1] + w[(i,j-1),(i,j)] & \text{if } j > 0. \end{cases}$$

- *w*[(*i* - 1, *j*), (*i*, *j*)] is weight of street between (*i* − 1, *j*) and (*i*, *j*)
- *w*[(*i*, *j* - 1), (*i*, *j*)] is weight of street between (*i*, *j* − 1) and (*i*, *j*)

MTP -- Solving recurrence using **dynamic programming**

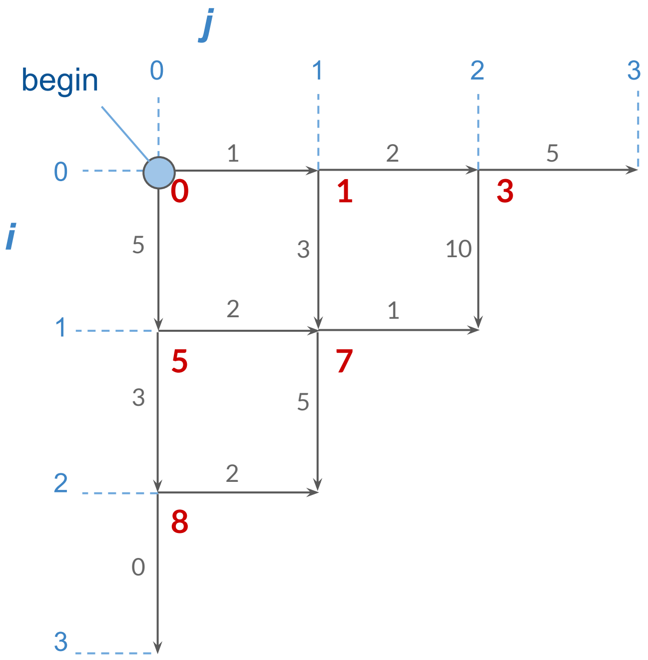


s[*i*, *j*] is the best score for path to coordinate (*i*, *j*)

$$s[i,j] = \max \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ s[i-1,j] + w[(i-1,j),(i,j)] & \text{if } i > 0, \\ s[i,j-1] + w[(i,j-1),(i,j)] & \text{if } j > 0. \end{cases}$$

- *w*[(*i* - 1, *j*), (*i*, *j*)] is weight of street between (*i* − 1, *j*) and (*i*, *j*)
- *w*[(*i*, *j* - 1), (*i*, *j*)] is weight of street between (*i*, *j* − 1) and (*i*, *j*)

MTP -- Solving recurrence using **dynamic programming**

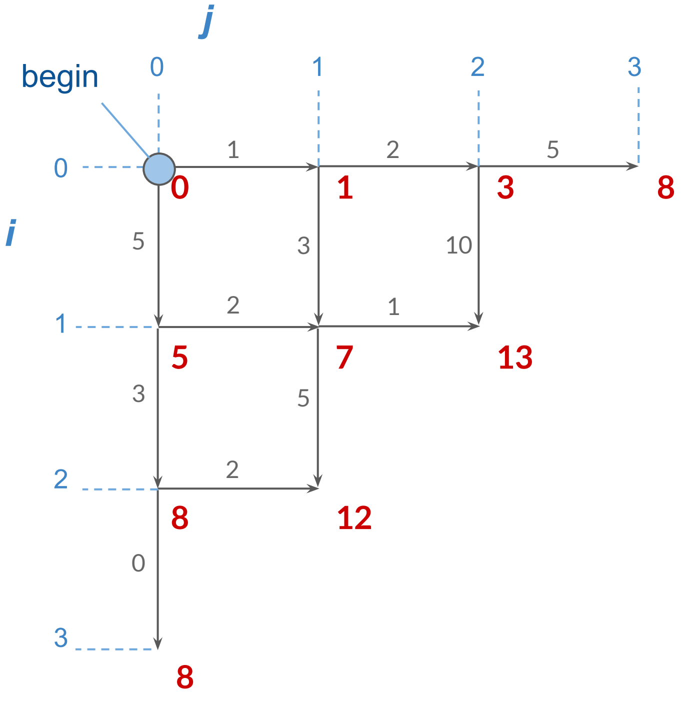


$s[i, j]$ is the best score for path to coordinate (i, j)

$$s[i,j] = \max \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ s[i-1,j] + w[(i-1,j),(i,j)] & \text{if } i > 0, \\ s[i,j-1] + w[(i,j-1),(i,j)] & \text{if } j > 0. \end{cases}$$

- $w[(i - 1, j), (i, j)]$ is weight of street between $(i - 1, j)$ and (i, j)
- $w[(i, j - 1), (i, j)]$ is weight of street between $(i, j - 1)$ and (i, j)

MTP -- Solving recurrence using **dynamic programming**

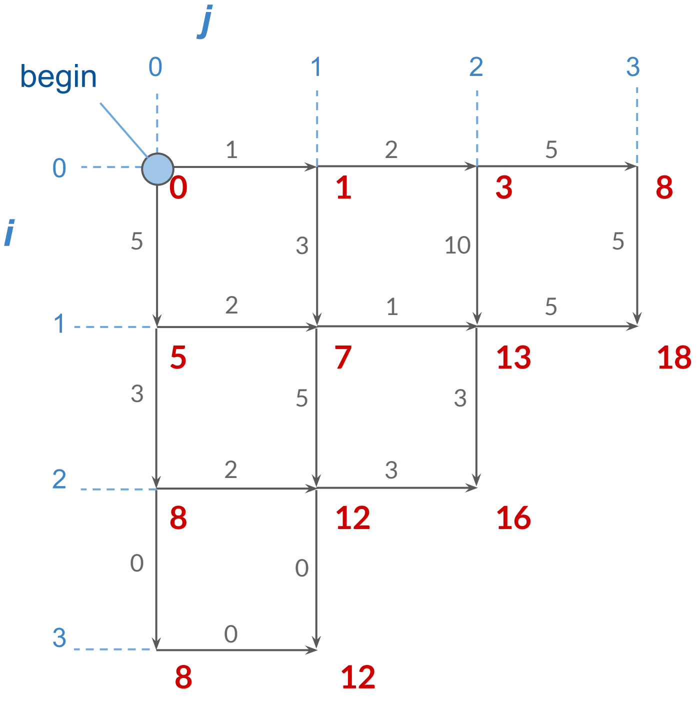


$s[i, j]$ is the best score for path to coordinate (i, j)

$$s[i,j] = \max \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ s[i-1,j] + w[(i-1,j),(i,j)] & \text{if } i > 0, \\ s[i,j-1] + w[(i,j-1),(i,j)] & \text{if } j > 0. \end{cases}$$

- $w[(i - 1, j), (i, j)]$ is weight of street between $(i - 1, j)$ and (i, j)
- $w[(i, j - 1), (i, j)]$ is weight of street between $(i, j - 1)$ and (i, j)

MTP -- Solving recurrence using **dynamic programming**

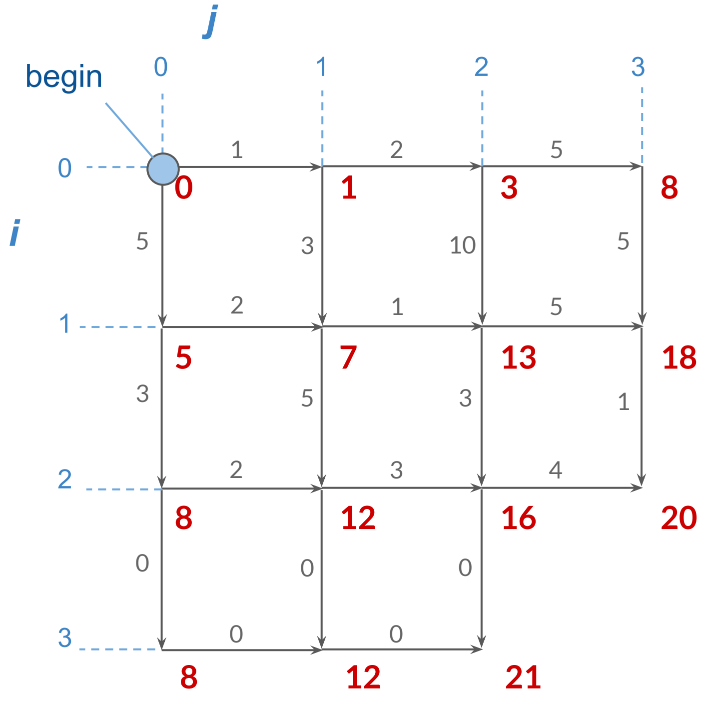


s[*i*, *j*] is the best score for path to coordinate (*i*, *j*)

$$s[i,j] = \max \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ s[i-1,j] + w[(i-1,j),(i,j)] & \text{if } i > 0, \\ s[i,j-1] + w[(i,j-1),(i,j)] & \text{if } j > 0. \end{cases}$$

- *w*[(*i* - 1, *j*), (*i*, *j*)] is weight of street between (*i* − 1, *j*) and (*i*, *j*)
- *w*[(*i*, *j* - 1), (*i*, *j*)] is weight of street between (*i*, *j* − 1) and (*i*, *j*)

MTP -- Solving recurrence using **dynamic programming**

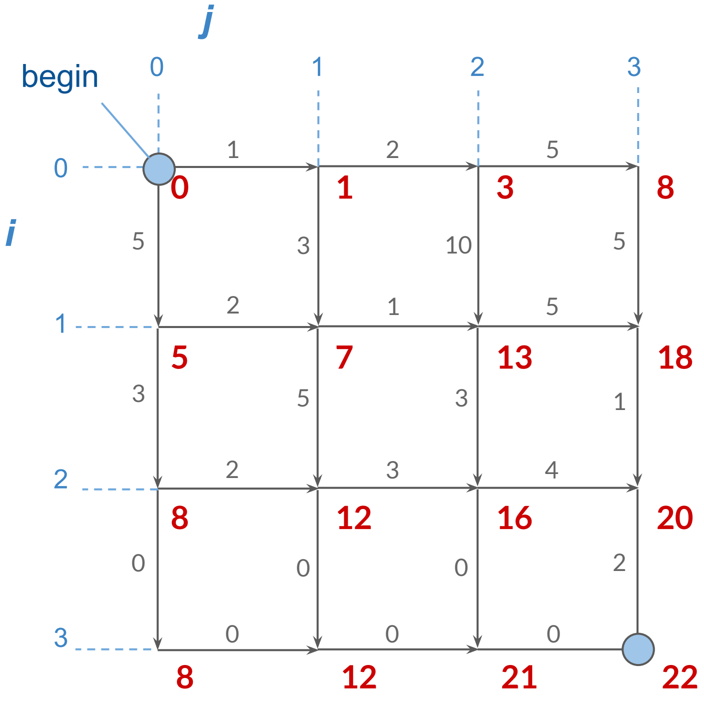


s[*i*, *j*] is the best score for path to coordinate (*i*, *j*)

$$s[i,j] = \max \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ s[i-1,j] + w[(i-1,j),(i,j)] & \text{if } i > 0, \\ s[i,j-1] + w[(i,j-1),(i,j)] & \text{if } j > 0. \end{cases}$$

- $w[(i - 1, j), (i, j)]$ is weight of street between $(i - 1, j)$ and (i, j)
- $w[(i, j - 1), (i, j)]$ is weight of street between $(i, j - 1)$ and (i, j)

MTP -- Solving recurrence using **dynamic programming**

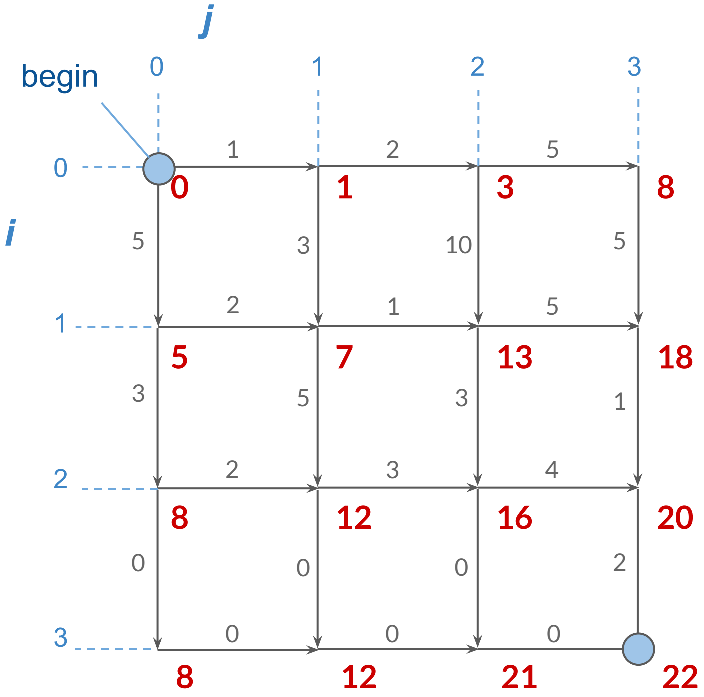


$s[i, j]$ is the best score for path to coordinate (i, j)

$$s[i,j] = \max \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ s[i-1,j] + w[(i-1,j),(i,j)] & \text{if } i > 0, \\ s[i,j-1] + w[(i,j-1),(i,j)] & \text{if } j > 0. \end{cases}$$

- $w[(i - 1, j), (i, j)]$ is weight of street between $(i - 1, j)$ and (i, j)
- $w[(i, j - 1), (i, j)]$ is weight of street between $(i, j - 1)$ and (i, j)

MTP -- Solving recurrence using **dynamic programming**



$s[i, j]$ is the best score for path to coordinate (i, j)

$$s[i,j] = \max \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ s[i-1,j] + w[(i-1,j),(i,j)] & \text{if } i > 0, \\ s[i,j-1] + w[(i,j-1),(i,j)] & \text{if } j > 0. \end{cases}$$

- $w[(i - 1, j), (i, j)]$ is weight of street between $(i - 1, j)$ and (i, j)
- $w[(i, j - 1), (i, j)]$ is weight of street between $(i, j - 1)$ and (i, j)

Let ***m*** be the number of rows and ***n*** be the number of columns.
Running time: $O(mn)$

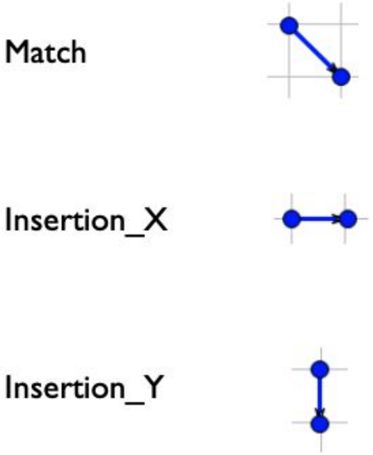
Recipe

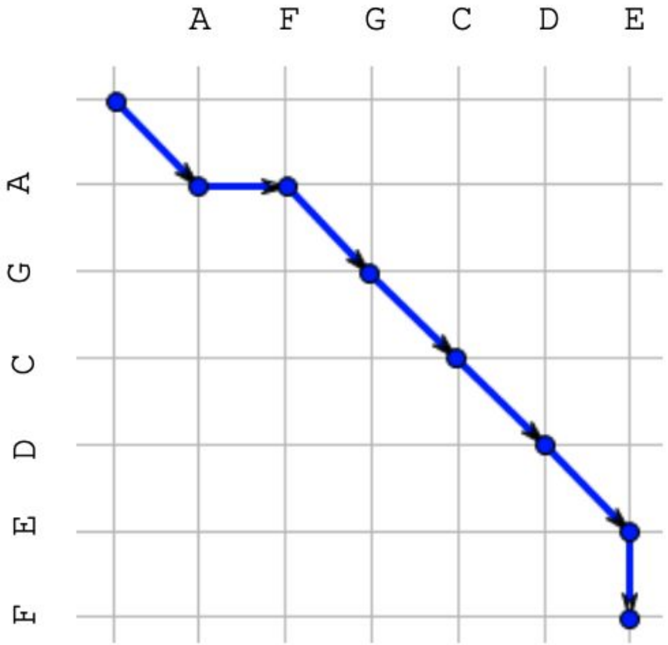
1. Identify subproblems

2. Write down recursions

3. Make it dynamic-programming!

The edit distance problem





A-GCDEF
AFGCDE-

Compute edit distance

Edit Distance problem: Given strings $\mathbf{v} \in \Sigma^m$ and $\mathbf{w} \in \Sigma^n$, compute the minimum number $d(\mathbf{v}, \mathbf{w})$ of elementary operations to transform $\mathbf{v}$ into $\mathbf{w}$.

v: `ATGTTAT...`
w: `AGCGTAC...`

deletion | insertion | mismatch | match

							$i-1$	i
prefix of **v** of length i	$\mathbf{v}_i$:	A	T	–	G	T	T	T
prefix of **w** of length j	$\mathbf{w}_j$:	A	G	C	G	T	–	C
							$j-1$	j

Optimal substructure:
Edit distance obtained from edit distance of prefix of string.

Compute edit distance - Optimal substructure

$d[i, j]$ is the edit distance of $\mathbf{v}_i$ and $\mathbf{w}_j$,
where $\mathbf{v}_i$ is prefix of $\mathbf{v}$ of length i and $\mathbf{w}_j$ is prefix of $\mathbf{w}$ of length j

Compute edit distance - Optimal substructure

$d[i, j]$ is the edit distance of $\mathbf{v}_i$ and $\mathbf{w}_j$,
where $\mathbf{v}_i$ is prefix of $\mathbf{v}$ of length i and $\mathbf{w}_j$ is prefix of $\mathbf{w}$ of length j

Deletion: $d[i, j] =$?
Extend by a character in $\mathbf{v}$

...	$\mathbf{v}_i$
...	–

Compute edit distance - Optimal substructure

$d[i, j]$ is the edit distance of $\mathbf{v}_i$ and $\mathbf{w}_j$,
where $\mathbf{v}_i$ is prefix of $\mathbf{v}$ of length i and $\mathbf{w}_j$ is prefix of $\mathbf{w}$ of length j

Deletion: $d[i, j] = d[i-1, j] + 1$
Extend by a character in $\mathbf{v}$

...	$\mathbf{v}_i$
...	–

Compute edit distance - Optimal substructure

$d[i, j]$ is the edit distance of $\mathbf{v}_i$ and $\mathbf{w}_j$,
where $\mathbf{v}_i$ is prefix of $\mathbf{v}$ of length i and $\mathbf{w}_j$ is prefix of $\mathbf{w}$ of length j

Deletion: $d[i, j] = d[i-1, j] + 1$
Extend by a character in $\mathbf{v}$

...	$\mathbf{v}_i$
...	–

Insertion: $d[i, j] =$?
Extend by a character in $\mathbf{w}$

...	–
...	$\mathbf{w}_j$

Compute edit distance - Optimal substructure

$d[i,j]$ is the edit distance of $\mathbf{v}_i$ and $\mathbf{w}_j$,
where $\mathbf{v}_i$ is prefix of $\mathbf{v}$ of length i and $\mathbf{w}_j$ is prefix of $\mathbf{w}$ of length j

Deletion: $d[i,j] = d[i-1,j] + 1$
Extend by a character in $\mathbf{v}$

...	$\mathbf{v}_i$
...	$-$

Insertion: $d[i,j] = d[i,j-1] + 1$
Extend by a character in $\mathbf{w}$

...	$-$
...	$\mathbf{w}_j$

Compute edit distance - Optimal substructure

$d[i,j]$ is the edit distance of $\mathbf{v}_i$ and $\mathbf{w}_j$,
where $\mathbf{v}_i$ is prefix of $\mathbf{v}$ of length i and $\mathbf{w}_j$ is prefix of $\mathbf{w}$ of length j

Deletion: $d[i,j] = d[i-1,j] + 1$
Extend by a character in $\mathbf{v}$

...	$\mathbf{v}_i$
...	–

Insertion: $d[i,j] = d[i,j-1] + 1$
Extend by a character in $\mathbf{w}$

...	–
...	$\mathbf{w}_j$

Mismatch: $d[i,j] =$?
Extend by a character in $\mathbf{v}$ and $\mathbf{w}$

...	$\mathbf{v}_i$
...	$\mathbf{w}_j$

Compute edit distance - Optimal substructure

$d[i,j]$ is the edit distance of $\mathbf{v}_i$ and $\mathbf{w}_j$,
where $\mathbf{v}_i$ is prefix of $\mathbf{v}$ of length i and $\mathbf{w}_j$ is prefix of $\mathbf{w}$ of length j

Deletion: $d[i,j] = d[i-1,j] + 1$
Extend by a character in **v**

...	$\mathbf{v}_i$
...	–

Insertion: $d[i,j] = d[i,j-1] + 1$
Extend by a character in **w**

...	–
...	$\mathbf{w}_j$

Mismatch: $d[i,j] = d[i-1,j-1] + 1$
Extend by a character in **v** and **w**

...	$\mathbf{v}_i$
...	$\mathbf{w}_j$

Compute edit distance - Optimal substructure

$d[i,j]$ is the edit distance of $\mathbf{v}_i$ and $\mathbf{w}_j$,
where $\mathbf{v}_i$ is prefix of $\mathbf{v}$ of length i and $\mathbf{w}_j$ is prefix of $\mathbf{w}$ of length j

Deletion: $d[i,j] = d[i-1,j] + 1$
Extend by a character in $\mathbf{v}$

...	$\mathbf{v}_i$
...	$-$

Insertion: $d[i,j] = d[i,j-1] + 1$
Extend by a character in $\mathbf{w}$

...	$-$
...	$\mathbf{w}_j$

Mismatch: $d[i,j] = d[i-1,j-1] + 1$
Extend by a character in $\mathbf{v}$ and $\mathbf{w}$

...	$\mathbf{v}_i$
...	$\mathbf{w}_j$

Match: $d[i,j] =$?
Extend by a character in $\mathbf{v}$ and $\mathbf{w}$

...	$\mathbf{v}_i$
...	$\mathbf{w}_j$

Compute edit distance - Optimal substructure

$d[i, j]$ is the edit distance of $\mathbf{v}_i$ and $\mathbf{w}_j$,
where $\mathbf{v}_i$ is prefix of $\mathbf{v}$ of length i and $\mathbf{w}_j$ is prefix of $\mathbf{w}$ of length j

Deletion: $d[i, j] = d[i-1, j] + 1$
Extend by a character in $\mathbf{v}$

...	$\mathbf{v}_i$
...	$-$

Insertion: $d[i, j] = d[i, j-1] + 1$
Extend by a character in $\mathbf{w}$

...	$-$
...	$\mathbf{w}_j$

Mismatch: $d[i, j] = d[i-1, j-1] + 1$
Extend by a character in $\mathbf{v}$ and $\mathbf{w}$

...	$\mathbf{v}_i$
...	$\mathbf{w}_j$

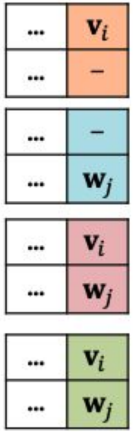
Match: $d[i, j] = d[i-1, j-1]$
Extend by a character in $\mathbf{v}$ and $\mathbf{w}$

...	$\mathbf{v}_i$
...	$\mathbf{w}_j$

Compute edit distance - Recurrence

$d[i, j]$ is the edit distance of $\mathbf{v}_i$ and $\mathbf{w}_j$,
where $\mathbf{v}_i$ is prefix of $\mathbf{v}$ of length i and $\mathbf{w}_j$ is prefix of $\mathbf{w}$ of length j

$$d[i,j] = \min \begin{cases} d[i-1, j] + 1, & \\ d[i, j-1] + 1, & \\ d[i-1, j-1] + 1, & \text{if } v_i \neq w_j, \\ d[i-1, j-1], & \text{if } v_i = w_j. \end{cases}$$



Compute edit distance - Recurrence

$d[i,j]$ is the edit distance of $\mathbf{v}_i$ and $\mathbf{w}_j$,
where $\mathbf{v}_i$ is prefix of $\mathbf{v}$ of length i and $\mathbf{w}_j$ is prefix of $\mathbf{w}$ of length j

$$d[i,j] = \min \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ d[i-1,j]+1, & \text{if } i > 0, \\ d[i,j-1]+1, & \text{if } j > 0, \\ d[i-1,j-1]+1, & \text{if } i > 0,\ j > 0 \text{ and } v_i \neq w_j, \\ d[i-1,j-1], & \text{if } i > 0,\ j > 0 \text{ and } v_i = w_j. \end{cases}$$

Compute edit distance - Dynamic Programming

Example:

w = ATCG
v = ATGT

$$d[i,j] = \min \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ d[i-1,j]+1, & \text{if } i > 0, \\ d[i,j-1]+1, & \text{if } j > 0, \\ d[i-1,j-1]+1, & \text{if } i > 0,\ j > 0 \text{ and } v_i \neq w_j, \\ d[i-1,j-1], & \text{if } i > 0,\ j > 0 \text{ and } v_i = w_j. \end{cases}$$

j		0	1	2	3	4
i	***v*** \ ***w***		A	T	C	G
0						
1	A					
2	T					
3	G					
4	T					

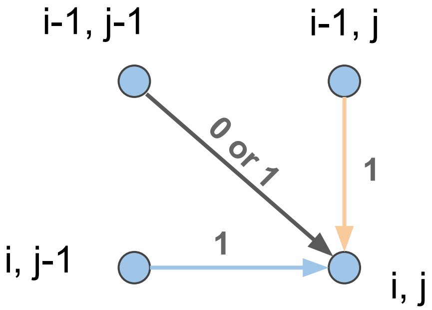
Compute edit distance - Dynamic Programming

Example:

w = ATCG
v = ATGT

	j	0	1	2	3	4
i	***v*** \ ***w***		A	T	C	G
0		0				
1	A					
2	T					
3	G					
4	T					

$$d[i,j] = \min \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ d[i-1,j] + 1, & \text{if } i > 0, \\ d[i,j-1] + 1, & \text{if } j > 0, \\ d[i-1,j-1] + 1, & \text{if } i > 0,\ j > 0 \text{ and } v_i \neq w_j, \\ d[i-1,j-1], & \text{if } i > 0,\ j > 0 \text{ and } v_i = w_j. \end{cases}$$



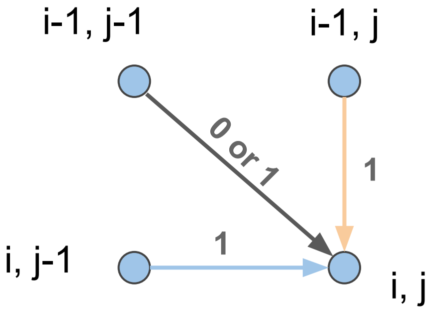
Compute edit distance - Dynamic Programming

Example:

w = ATCG
v = ATGT

i \ ***j***		0	1	2	3	4
	v \ ***w***		A	T	C	G
0		0	1	2	3	4
1	A	1				
2	T	2				
3	G	3				
4	T	4				

$$d[i,j] = \min \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ d[i-1,j]+1, & \text{if } i > 0, \\ d[i,j-1]+1, & \text{if } j > 0, \\ d[i-1,j-1]+1, & \text{if } i > 0,\ j > 0 \text{ and } v_i \neq w_j, \\ d[i-1,j-1], & \text{if } i > 0,\ j > 0 \text{ and } v_i = w_j. \end{cases}$$



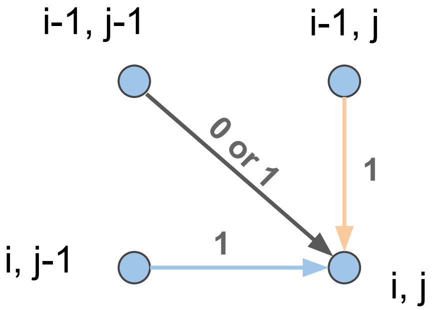
Compute edit distance - Dynamic Programming

Example:

w = ATCG
v = ATGT

$$d[i,j] = \min \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ d[i-1,j]+1, & \text{if } i > 0, \\ d[i,j-1]+1, & \text{if } j > 0, \\ d[i-1,j-1]+1, & \text{if } i > 0,\ j > 0 \text{ and } v_i \neq w_j, \\ d[i-1,j-1], & \text{if } i > 0,\ j > 0 \text{ and } v_i = w_j. \end{cases}$$

i \ ***j***		0	1	2	3	4
	v \ ***w***		A	T	C	G
0		0	1	2	3	4
1	A	1	?			
2	T	2				
3	G	3				
4	T	4				



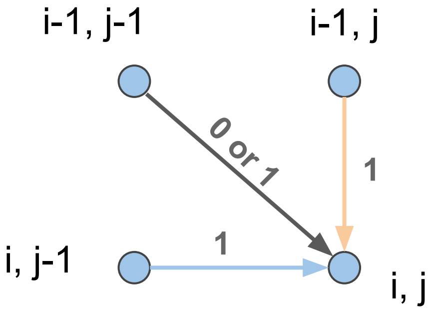
Compute edit distance - Dynamic Programming

Example:

w = ATCG
v = ATGT

$$d[i,j] = \min \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ d[i-1,j]+1, & \text{if } i > 0, \\ d[i,j-1]+1, & \text{if } j > 0, \\ d[i-1,j-1]+1, & \text{if } i > 0,\ j > 0 \text{ and } v_i \neq w_j, \\ d[i-1,j-1], & \text{if } i > 0,\ j > 0 \text{ and } v_i = w_j. \end{cases}$$

i \ *j*		0	1	2	3	4
	v \ ***w***		A	T	C	G
0		0	1	2	3	4
1	A	1	0			
2	T	2				
3	G	3				
4	T	4				



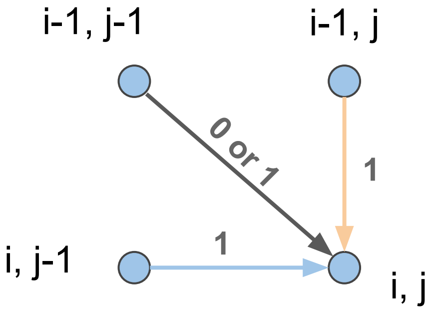
Compute edit distance - Dynamic Programming

Example:

w = ATCG
v = ATGT

i \ ***j***	***v*** \ ***w***	0	1	2	3	4
			A	T	C	G
0		0	1	2	3	4
1	A	1	0			
2	T	2	?			
3	G	3				
4	T	4				

$$d[i,j] = \min \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ d[i-1,j]+1, & \text{if } i > 0, \\ d[i,j-1]+1, & \text{if } j > 0, \\ d[i-1,j-1]+1, & \text{if } i > 0,\ j > 0 \text{ and } v_i \neq w_j, \\ d[i-1,j-1], & \text{if } i > 0,\ j > 0 \text{ and } v_i = w_j. \end{cases}$$



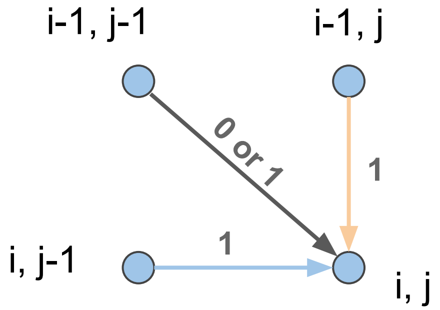
Compute edit distance - Dynamic Programming

Example:

w = ATCG
v = ATGT

$$
d[i,j] = \min \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ d[i-1,j]+1, & \text{if } i > 0, \\ d[i,j-1]+1, & \text{if } j > 0, \\ d[i-1,j-1]+1, & \text{if } i > 0,\ j > 0 \text{ and } v_i \neq w_j, \\ d[i-1,j-1], & \text{if } i > 0,\ j > 0 \text{ and } v_i = w_j. \end{cases}
$$

	j	0	1	2	3	4
i	***w*** / ***v***		A	T	C	G
0		0	1	2	3	4
1	A	1	0			
2	T	2	1			
3	G	3				
4	T	4				



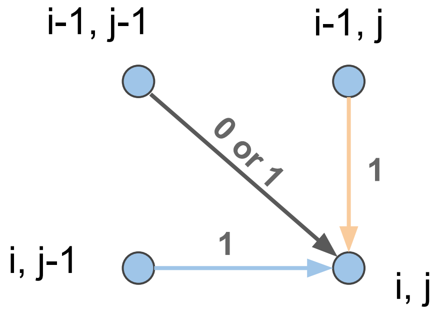
Compute edit distance - Dynamic Programming

Example:

w = ATCG
v = ATGT

$$d[i,j] = \min \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ d[i-1,j]+1, & \text{if } i > 0, \\ d[i,j-1]+1, & \text{if } j > 0, \\ d[i-1,j-1]+1, & \text{if } i > 0,\ j > 0 \text{ and } v_i \neq w_j, \\ d[i-1,j-1], & \text{if } i > 0,\ j > 0 \text{ and } v_i = w_j. \end{cases}$$

	j	0	1	2	3	4
i	***v*** \ ***w***		A	T	C	G
0		0	1	2	3	4
1	A	1				
2	T	2				
3	G	3				
4	T	4				



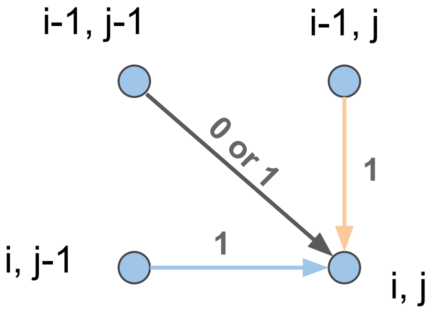
Compute edit distance - Dynamic Programming

Example:

w = ATCG
v = ATGT

$$d[i,j] = \min \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ d[i-1,j] + 1, & \text{if } i > 0, \\ d[i,j-1] + 1, & \text{if } j > 0, \\ d[i-1,j-1] + 1, & \text{if } i > 0,\ j > 0 \text{ and } v_i \neq w_j, \\ d[i-1,j-1], & \text{if } i > 0,\ j > 0 \text{ and } v_i = w_j. \end{cases}$$

i \ *j*	*v* \ *w*	0	1	2	3	4
			A	T	C	G
0		0	1	2	3	4
1	A	1				
2	T	2				
3	G	3				
4	T	4				



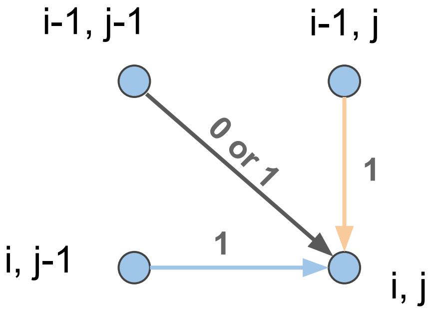
Compute edit distance - Dynamic Programming

Example:

w = ATCG
v = ATGT

$$d[i,j] = \min \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ d[i-1,j]+1, & \text{if } i > 0, \\ d[i,j-1]+1, & \text{if } j > 0, \\ d[i-1,j-1]+1, & \text{if } i > 0,\ j > 0 \text{ and } v_i \neq w_j, \\ d[i-1,j-1], & \text{if } i > 0,\ j > 0 \text{ and } v_i = w_j. \end{cases}$$

	j	0	1	2	3	4
i	***v*** \ ***w***		A	T	C	G
0		0	1	2	3	4
1	A	1				
2	T	2				
3	G	3				
4	T	4				



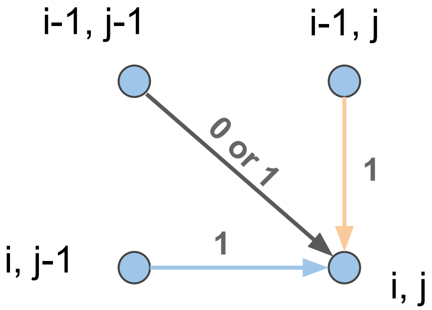
Compute edit distance - Dynamic Programming

Example:

w = ATCG
v = ATGT

$$d[i,j] = \min \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ d[i-1,j]+1, & \text{if } i > 0, \\ d[i,j-1]+1, & \text{if } j > 0, \\ d[i-1,j-1]+1, & \text{if } i > 0,\ j > 0 \text{ and } v_i \neq w_j, \\ d[i-1,j-1], & \text{if } i > 0,\ j > 0 \text{ and } v_i = w_j. \end{cases}$$

	j	0	1	2	3	4
i	***v* \ *w***		A	T	C	G
0		0	1	2	3	4
1	A	1	0	1	2	3
2	T	2	1	0	1	2
3	G	3	2	1	1	1
4	T	4	3	2	2	2

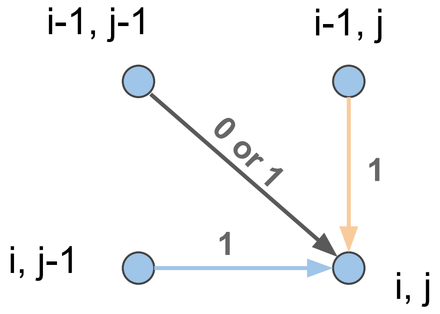


Output the alignment -- Finding optimal path(s)

Key idea: where does the score of the current cell come from?

$$d[i,j] = \min \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ \boxed{d[i-1,j]+1,} & \text{if } i > 0, \\ d[i,j-1]+1, & \text{if } j > 0, \\ \boxed{d[i-1,j-1]+1,} & \text{if } i > 0,\ j > 0 \text{ and } v_i \neq w_j, \\ d[i-1,j-1], & \text{if } i > 0,\ j > 0 \text{ and } v_i = w_j. \end{cases}$$

i \ *j*		0	1	2	3	4
	v \ *w*		A	T	C	G
0		0	1	2	3	4
1	A	1	0	1	2	3
2	T	2	1	0	1	2
3	G	3	2	1	1	1
4	T	4	3	2	2	2

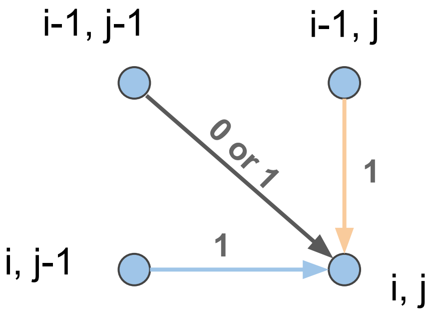


Output the alignment -- Finding optimal path(s)

Key idea: where does the score of the current cell come from?

$$
d[i,j] = \min \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ d[i-1,j] + 1, & \text{if } i > 0, \\ d[i,j-1] + 1, & \text{if } j > 0, \\ d[i-1,j-1] + 1, & \text{if } i > 0,\ j > 0 \text{ and } v_i \neq w_j, \\ d[i-1,j-1], & \text{if } i > 0,\ j > 0 \text{ and } v_i = w_j. \end{cases}
$$

	j	0	1	2	3	4
i	*v* \ *w*		A	T	C	G
0		0	1	2	3	4
1	A	1	0	1	2	3
2	T	2	1	0	1	2
3	G	3	2	1	1	1
4	T	4	3	2	2	2

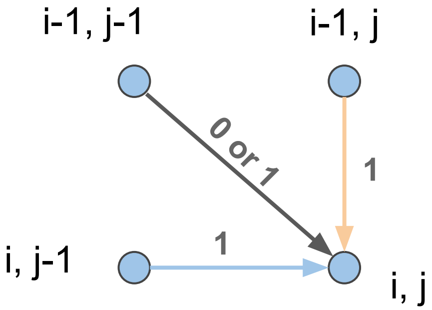


Output the alignment -- Finding optimal path(s)

Key idea: where does the score of the current cell come from?

$$d[i,j] = \min \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ d[i-1,j]+1, & \text{if } i > 0, \\ d[i,j-1]+1, & \text{if } j > 0, \\ d[i-1,j-1]+1, & \text{if } i > 0,\ j > 0 \text{ and } v_i \neq w_j, \\ \boxed{d[i-1,j-1],} & \text{if } i > 0,\ j > 0 \text{ and } v_i = w_j. \end{cases}$$

	j	0	1	2	3	4
i	*v* \ *w*		A	T	C	G
0		0	1	2	3	4
1	A	1	0	1	2	3
2	T	2	1	0	1	2
3	G	3	2	1	1	1
4	T	4	3	2	2	2

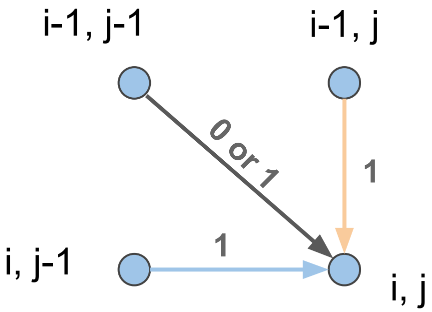


Output the alignment -- Finding optimal path(s)

Key idea: where does the score of the current cell come from?

$$d[i,j] = \min \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ d[i-1,j]+1, & \text{if } i > 0, \\ d[i,j-1]+1, & \text{if } j > 0, \\ d[i-1,j-1]+1, & \text{if } i > 0,\ j > 0 \text{ and } v_i \neq w_j, \\ \boxed{d[i-1,j-1],} & \text{if } i > 0,\ j > 0 \text{ and } v_i = w_j. \end{cases}$$

i \ *j*		0	1	2	3	4
	v \ *w*		A	T	C	G
0		0	1	2	3	4
1	A	1	0	1	2	3
2	T	2	1	0	1	2
3	G	3	2	1	1	1
4	T	4	3	2	2	2

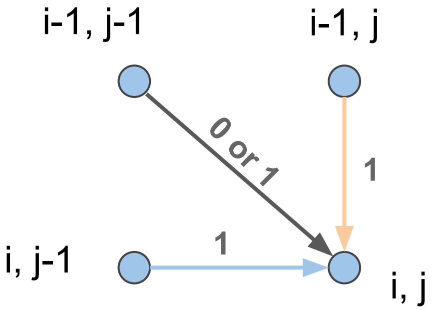


Output the alignment -- Finding optimal path(s)

Key idea: where does the score of the current cell come from?

$$d[i,j] = \min \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ d[i-1,j]+1, & \text{if } i > 0, \\ d[i,j-1]+1, & \text{if } j > 0, \\ d[i-1,j-1]+1, & \text{if } i > 0,\ j > 0 \text{ and } v_i \neq w_j, \\ d[i-1,j-1], & \text{if } i > 0,\ j > 0 \text{ and } v_i = w_j. \end{cases}$$

i \ *j*		0	1	2	3	4
	v \ *w*		A	T	C	G
0		0	1	2	3	4
1	A	1	0	1	2	3
2	T	2	1	0	1	2
3	G	3	2	1	1	1
4	T	4	3	2	2	2



Backtrace algorithm

- Base conditions:

  `D(i,0) = i`   `D(0,j) = j`

- Termination:

  `D(N,M) is distance`

- Recurrence Relation:

```
For each  i = 1…M
      For each  j = 1…N
```

$$D(i,j) = \min \begin{cases} D(i-1,j) + 1 & \text{deletion} \\ D(i,j-1) + 1 & \text{insertion} \\ D(i-1,j-1) + \begin{cases} 1; & \text{if } X(i) \neq Y(j) \\ 0; & \text{if } X(i) = Y(j) \end{cases} & \begin{matrix} \text{substitution} \\ \text{match} \end{matrix} \end{cases}$$

$$ptr(i,j) = \begin{cases} \text{LEFT} & \text{insertion} \\ \text{DOWN} & \text{deletion} \\ \text{DIAG} & \text{substitution} \quad \text{match} \end{cases}$$

Output the alignment

Key idea: where does the score of the current cell come from?

$$d[i,j] = \min \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ d[i-1,j] + 1, & \text{if } i > 0, \\ d[i,j-1] + 1, & \text{if } j > 0, \\ d[i-1,j-1] + 1, & \text{if } i > 0,\ j > 0 \text{ and } v_i \neq w_j, \\ d[i-1,j-1], & \text{if } i > 0,\ j > 0 \text{ and } v_i = w_j. \end{cases}$$

i \ *j*		0	1	2	3	4
	v \ *w*		A	T	C	G
0		0	1	2	3	4
1	A	1	0	1	2	3
2	T	2	1	0	1	2
3	G	3	2	1	1	1
4	T	4	3	2	2	2

A	T	–	G	T
A	T	C	G	–

A	T	G	T
A	T	C	G

Computing edit distance -- Running time

$\begin{array}{|c|c|}\hline \cdots & v_i \\ \hline \cdots & - \\ \hline\end{array}$ deletion $\begin{array}{|c|c|}\hline \cdots & - \\ \hline \cdots & w_j \\ \hline\end{array}$ insertion $\begin{array}{|c|c|}\hline \cdots & v_i \\ \hline \cdots & w_j \\ \hline\end{array}$ mismatch $\begin{array}{|c|c|}\hline \cdots & v_i \\ \hline \cdots & w_j \\ \hline\end{array}$ match

$$d[i,j] = \min \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ d[i-1,j]+1, & \text{if } i > 0, \\ d[i,j-1]+1, & \text{if } j > 0, \\ d[i-1,j-1]+1, & \text{if } i > 0,\ j > 0 \text{ and } v_i \neq w_j, \\ d[i-1,j-1], & \text{if } i > 0,\ j > 0 \text{ and } v_i = w_j. \end{cases}$$

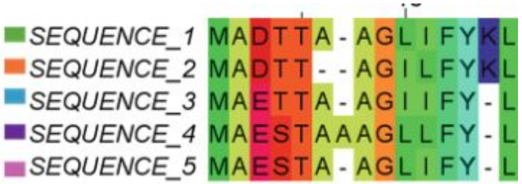
j		0	1	2	3	4
i	***v* \ *w***		A	T	C	G
0		0	1	2	3	4
1	A	1	0	1	2	3
2	T	2	1	0	1	2
3	G	3	2	1	1	1
4	T	4	3	2	2	2

For each $(m+1)\times(n+1)$ entry:
- 3 addition operations
- 1 comparison operation
- 1 minimum operation

Running time: $O(mn)$ time

Conclusions

- Sequence alignment



- Algorithm for alignment
  - Edit distance
  - Dynamic programming

$$
d[i,j] = \min \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ d[i-1,j] + 1, & \text{if } i > 0, \\ d[i,j-1] + 1, & \text{if } j > 0, \\ d[i-1,j-1] + 1, & \text{if } i > 0,\ j > 0 \text{ and } v_i \neq w_j, \\ d[i-1,j-1], & \text{if } i > 0,\ j > 0 \text{ and } v_i = w_j. \end{cases}
$$

- Find the optimal alignment
  - Backtrace

j		0	1	2	3	4
i	*v* \ *w*		A	T	C	G
0		0	1	2	3	4
1	A	1	0	1	2	3
2	T	2	1	0	1	2
3	G	3	2	1	1	1
4	T	4	3	2	2	2