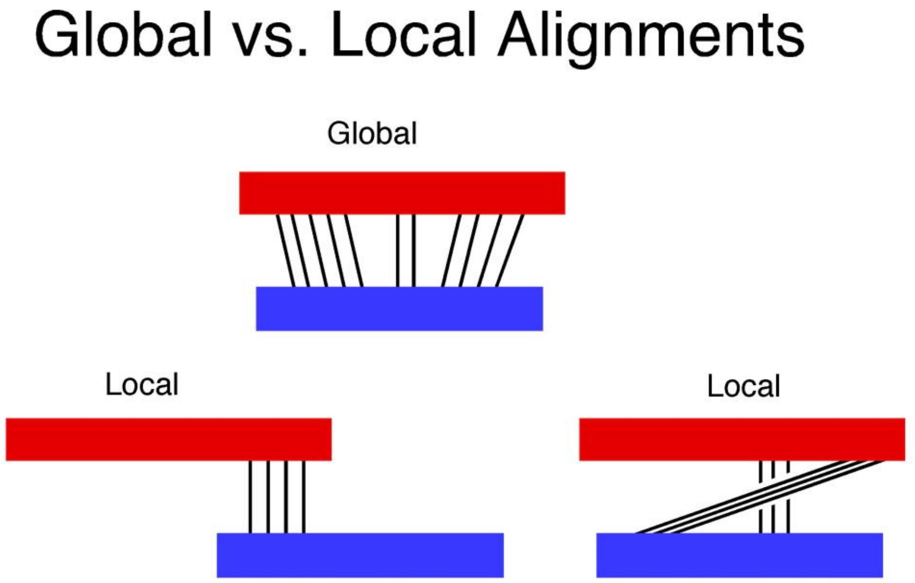
EECS730: Introduction to Bioinformatics

Lecture 04: Variations of sequence alignments

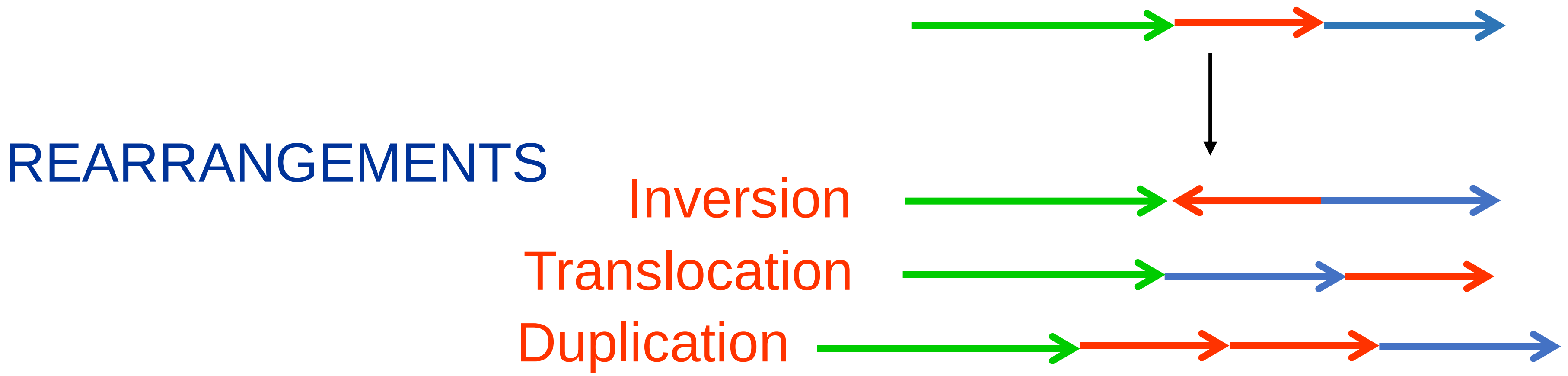


http://www.pitt.edu/~mcs2/teaching/biocomp/tutorials/global.html

Slides adapted from Dr. Shaojie Zhang (University of Central Florida)

Global alignment vs Local alignment

- Genome rearrangement usually shuffles the genome



- Protein domains have relatively well-annotated functions
- Similar for non-coding RNAs

Global alignment vs Local alignment

- Global Alignment

```
--T—-CC-C-AGT—-TATGT-CAGGGGACACG—A-GCATGCAGA-GAC
  |  || |  ||  | | | |||    || | | |  | ||||   |
AATTGCCGCC-GTCGT-T-TTCAG----CA-GTTATG—T-CAGAT--C
```

- Local Alignment—better alignment to find conserved segment

```
                  tccCAGTTATGTCAGgggacacgagcatgcagagac
                     ||||||||||||
aattgccgccgtcgttttcagCAGTTATGTCAGatc
```

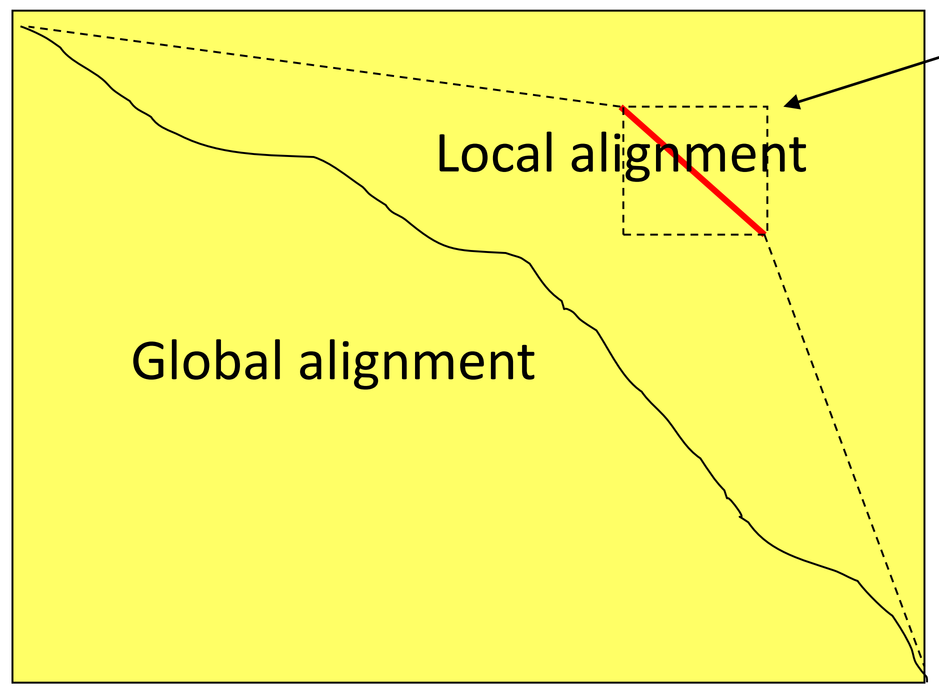
Global alignment vs Local alignment

- The Global Alignment Problem tries to find the longest path between vertices *(0,0)* and (*n,m*) in the edit graph.

- The Local Alignment Problem tries to find the longest path among paths between **arbitrary vertices** (*i,j*) and (*i', j'*) in the edit graph.

- Local alignment usually require less edit operations than Global alignment

Local alignment example



Compute a "mini" Global Alignment to get Local

Local alignment problem formulation (edit distance)

- Goal: Find the best local alignment between two strings
- Input : Strings **v, w**
- Output : Alignment of substrings of **v** and **w** whose number of edit operations is minimized

Can you see the problem of the formulation???

Local alignment problem formulation

- Empty substrings will always have an <u>**edit distance of 0**</u>! So they are optimal but meaningless!!!

- Since we have the “cost”, let’s also define “gain”!

- If we match two identical characters, we <u>**gain**</u> some information!

Local alignment problem formulation

- Goal: Find the best local alignment between two strings
- Input : Strings **v, w,** some **gain function** for matching identical characters and some **cost function** for matching different characters or opening gaps
- Output : Alignment of substrings of **v** and **w** with **maximized "profit"**

Local Alignment

- Naïve running time O(n^6)!!!

- For each combination of <*i, j, i', j'*>, perform global alignment!!!
- There are O(n^4) different combinations, each combination requires O(n^2) global alignment, totaling to O(n^6) running time!!!

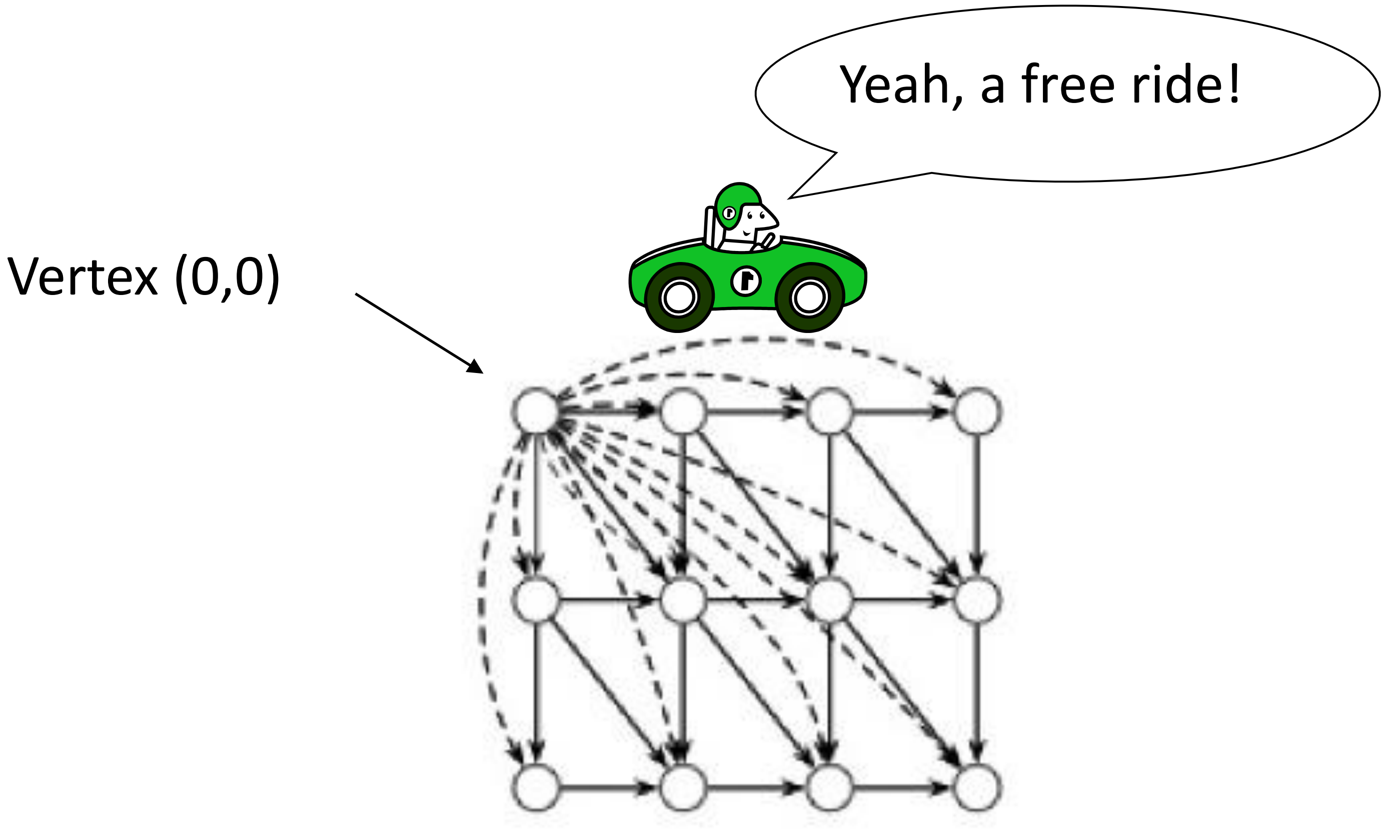
- **We can reduce that to O(n^4), how???**

Local alignment

- Notice that in the DP table, entry (i, j) stores the optimal alignment computed for substring (0, i) an (0, j).
- It means that for each pair of (i, i’), performing O(n^2) alignment would give us solutions for O(n^2) substrings as well!!!
- So we only need to try all possible <i, i’> combinations, which drops the total time complexity to O(n^4)
- **But it is still not satisfying…**

Smith-Waterman algorithm

The idea is that we only want to look at "good alignments"; of an alignment is "bad", we should be able to initialize a new alignment for free



The dashed edges represent the free rides from (0,0) to every other node.

Smith-Waterman alignment

- The recurrence:

$$s_{i,j} = \max \begin{cases} 0 \\ s_{i-1,j-1} + \delta(v_i, w_j) \\ s_{i-1,j} + \delta(v_i, -) \\ s_{i,j-1} + \delta(-, w_j) \end{cases}$$

Power of ZERO: there is only this change from the original recurrence of a Global Alignment - since there is only one “free ride” edge entering into every vertex

*Smith, Temple F. & Waterman, Michael S. (1981). "Identification of Common Molecular Subsequences". Journal of Molecular Biology. **147**: 195–197. doi:10.1016/0022-2836(81)90087-5. PMID 7265238.*

On implementation details

- Matrix initialization: Since there are free rides, we should initialize the first column and the first row to all 0s

v \ w			A	T	C	G	T	A	C
		0	1	2	3	4	5	6	7
	0	0	0	0	0	0	0	0	0
A	1	0							
T	2	0							
G	3	0							
T	4	0							
T	5	0							
A	6	0							
T	7	0							

Fill the table with the new recursive function with the magic 0

On implementation details

- In additional to the best path, we also need to note the termination of an alignment segment in the trace-back matrix
- The optimal local alignment score can always be found at the entry with the highest alignment “profit”

Semi-global and semi-local alignment

- Given two sequences v and w, in many cases we are want to align the entire sequence of v to a substring of w.
- For example, if v is a gene and w is a genome and we want to find the homolog of v in w. Note that using local alignment would detect many domains; and we want to make sure the entire sequence of v is aligned.
- Or If v is a gene and w is a sequencing fragment (read), and we want to know whether w is sampled from v. In this case we want to compare the entire sequence of w.
- **How to we modify the current algorithm to perform semi-global/semi-local alignments?**

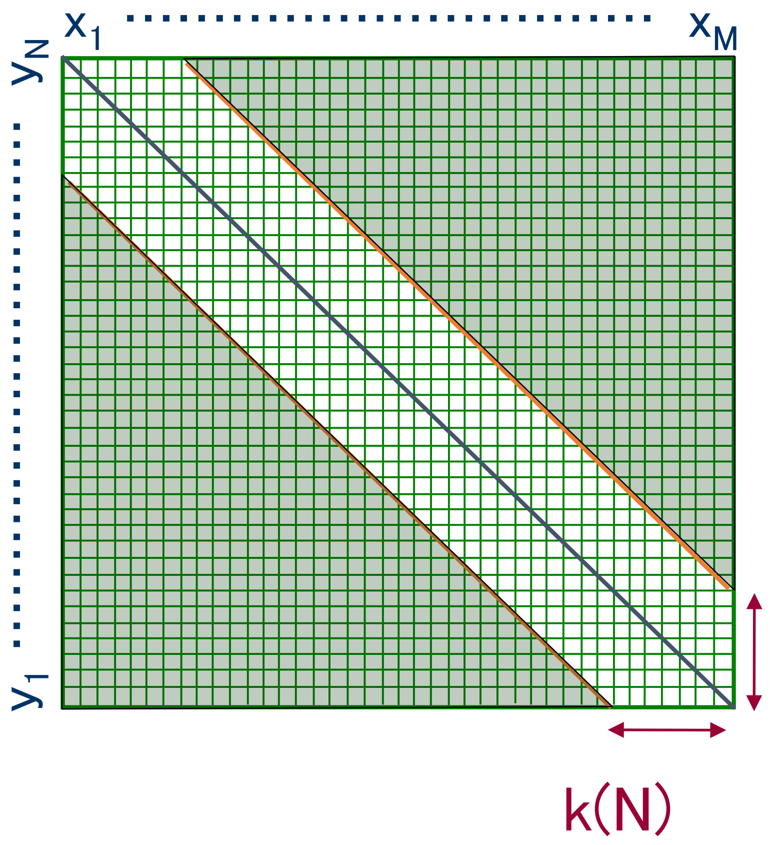
Semi-global and semi-local alignment

- Modifying the initialization and trace back of the global alignment algorithm
  - Free rides to entries in the first column/first row
  - identifying the highest “profit” in the last column/last row

Banding

- Quadratic time solution is still too slow
- The average gene length of human is ~15K bp long; aligning two genes would need to fill up ~225M DP entries.
- Intuition: **we are interested in “good” alignments rather than “bad” alignments**; and “good” alignments usually contain fewer gaps because gaps trigger “cost” instead of “gain”
- In the alignment table, less gap means that the path is located at the diagonal of the table

Banding cont.



Initialization:

F(i,0), F(0,j) undefined for i, j > k

Iteration:

```
For i = 1…N
  For j = max(1, i – k)…min(N, i+k)
```

$$F(i,j) = \max \begin{cases} F(i-1, j-1) + s(x_i, y_j) \\ F(i, j-1) - d, \text{ if } i - j > k(N) \\ F(i-1, j) - d, \text{ if } j - i > k(N) \end{cases}$$

Termination: same

Time complexity reduced to linear because the band size is considered as a constant!!!

Caveats on the use of banding

- **Banding is not a correct algorithm!**
  - Because an optimal path can pass through regions outside the banded region

- **Not with local alignments!** Because “good” local paths do not necessarily locate on the diagonal

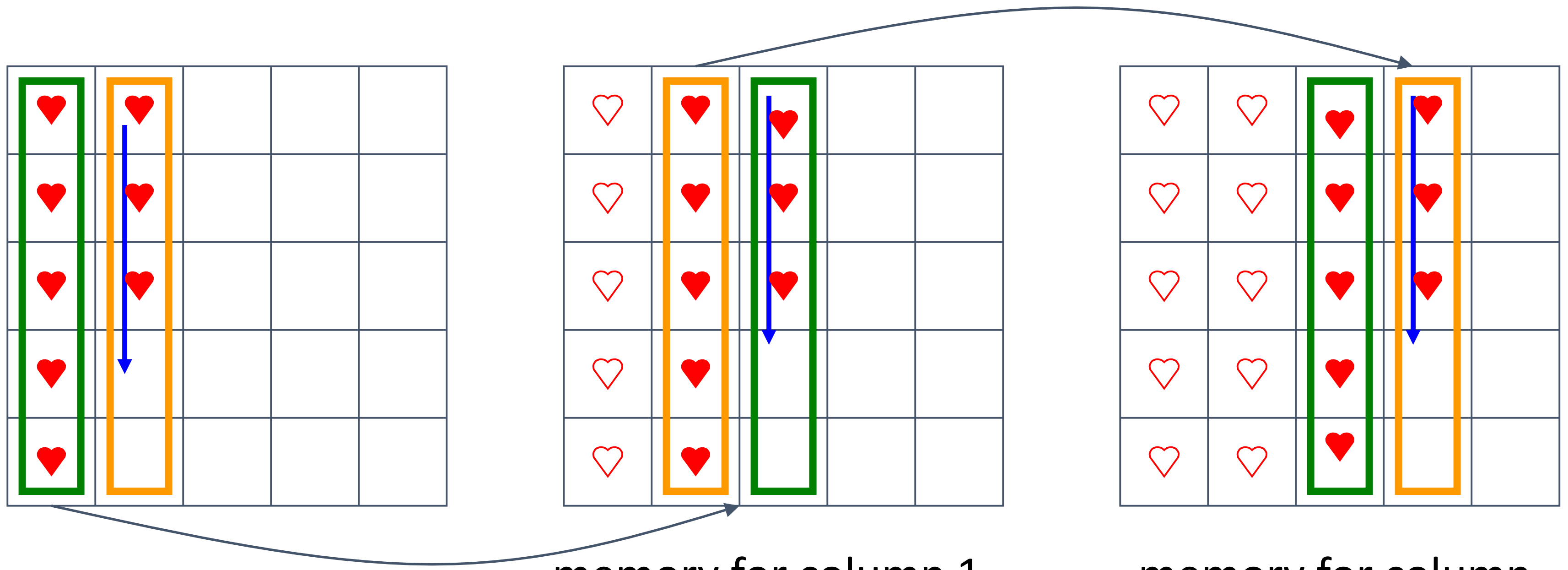
- For “asymmetric” global alignment (one sequence is much longer/shorter than the other) the banding should also be set asymmetrically
  - One dimension of size b
  - The other dimension of size abs(|w| - |v|)
  - The total number of entries to be filled is abs(|w| - |v|) * k(N)

Linear space global alignment

- Imagine that we are globally aligning two bacterial genomes that are ~3M long each; and we want to find the optimal answer so we do not want to use banding.
- Roughly speaking, 3M * 3M = 9000G
- It takes approximately 9000 secs to find the answer (provided that the CPU has a frequency of several GHz); 9000 secs is approximately 3hrs, which is OK.
- The real problem is to find a machine with 9000G/9T memory……
- Myers, G. and Miller, W., Optimal alignments in Linear Space, *Comput Appl Biosci* (1988) 4 (1): 11-17. doi: 10.1093/bioinformatics/4.1.11

Linear space solution

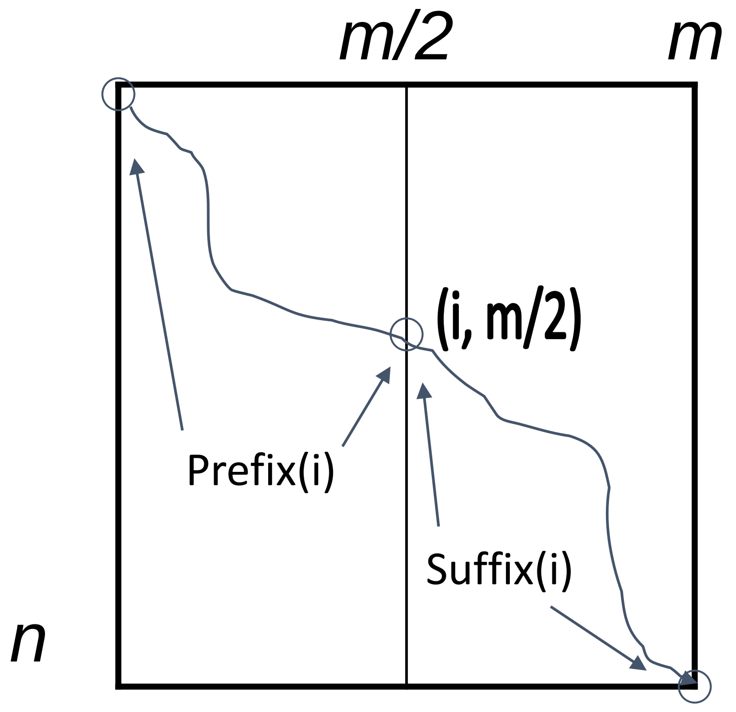
- The need for quadratic space is to facilitate trace back; without track back (such that we only know the “profit”), a simple change is capable of reducing the space to linear.



memory for column 1 is used to calculate column 3

memory for column 2 is used to calculate column 4

Linear space solution



Observation: alignment between (i, j) and (i', j') is equivalent to the alignment of their reversed strings , i.e. between (j, i) and (j', i')

We want to calculate the longest path from (0,0) to (n,m) that passes through (i,m/2) where i ranges from 0 to n and represents the i-th row

Define

length(i)

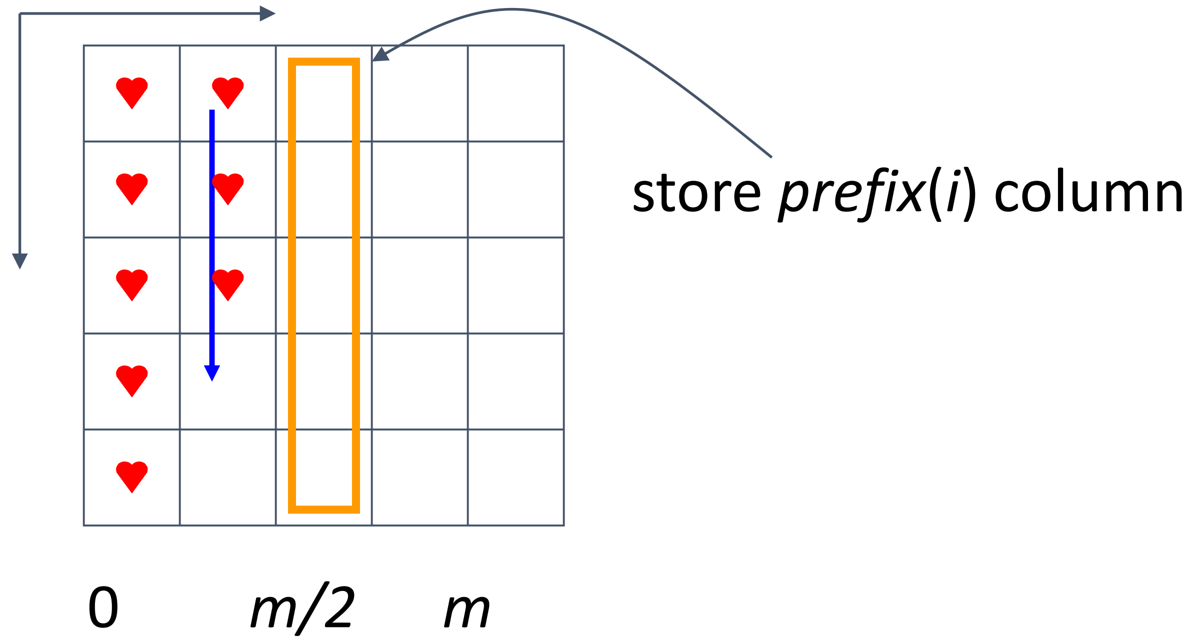
as the length of the longest path from (0,0) to (n,m) that passes through vertex (i, m/2)

Linear space solution

- We know that the path has to pass column m/2
- Optimal alignment computed between (n, k) and (m, m/2) is equivalent to the optimal alignment computed between (k, n) and (m/2, m)
- The optimal alignment computed between (0, 0) and (n, m) thus corresponds to the maximal sum of profits between (0, k), (0, m/2) and (n, k), (m, m/2) for all 0 <= k <= n
- Finding k is trivial and takes linear time

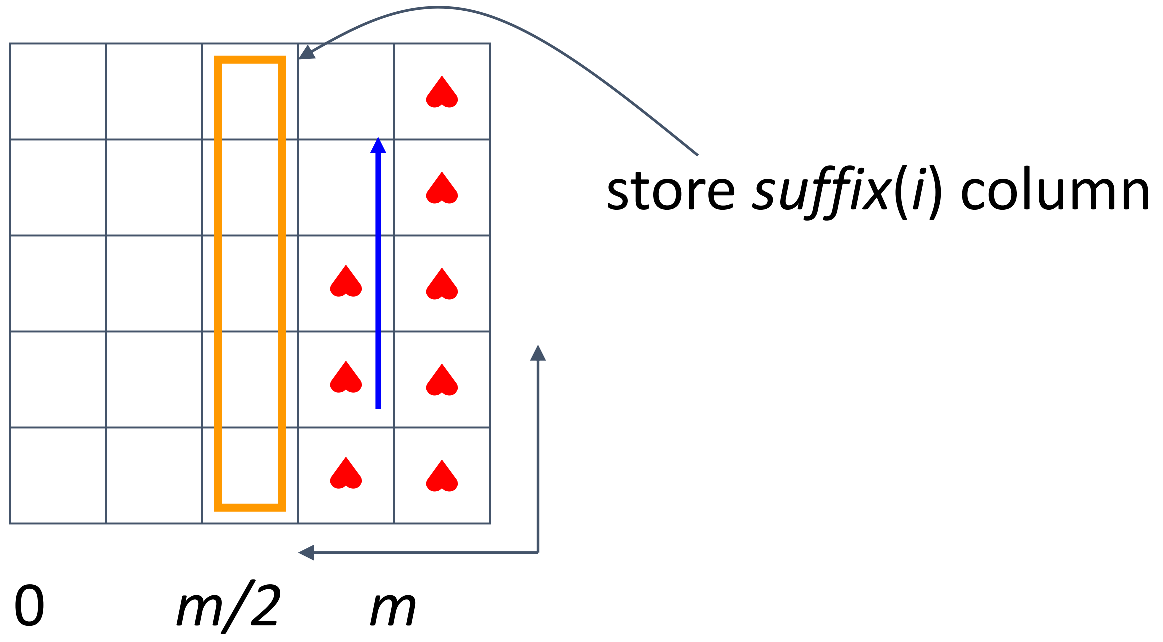
Computing prefix

- *prefix*(*i*) is the length of the longest path from (0,0) to (*i*,*m*/2)
- Compute *prefix*(*i*) by dynamic programming in the left half of the matrix



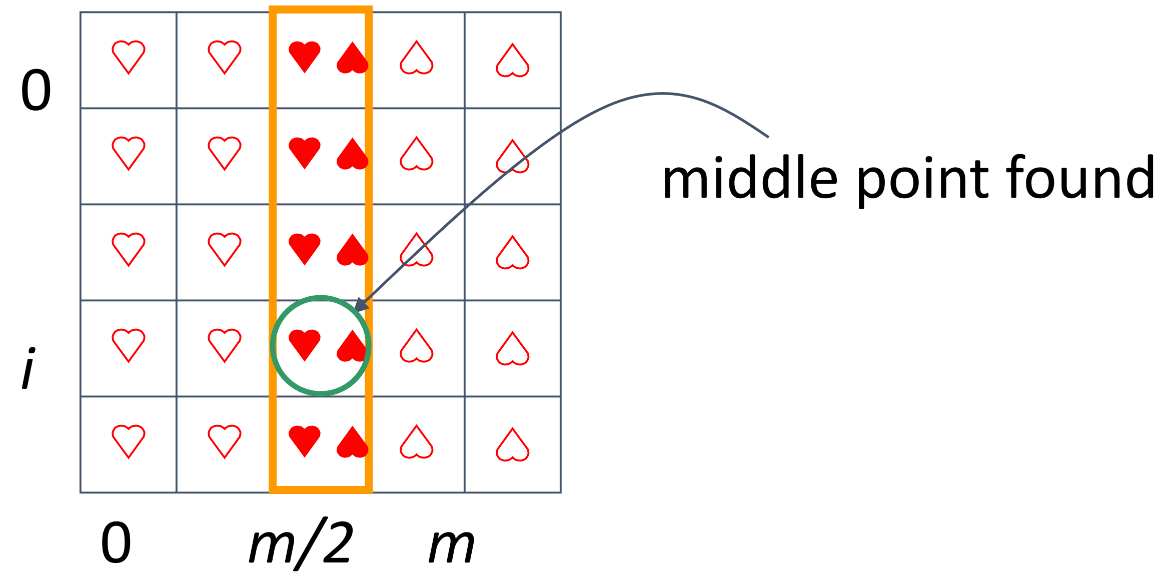
Computing the suffix

- *suffix*(*i*) is the length of the longest path from (*i*,*m*/2) to *(n,m)*
- *suffix*(*i*) is the length of the longest path from (*n*,*m*) to (*i*,*m*/2) with all edges reversed
- Compute *suffix*(*i*) by dynamic programming in the right half of the “reversed” matrix

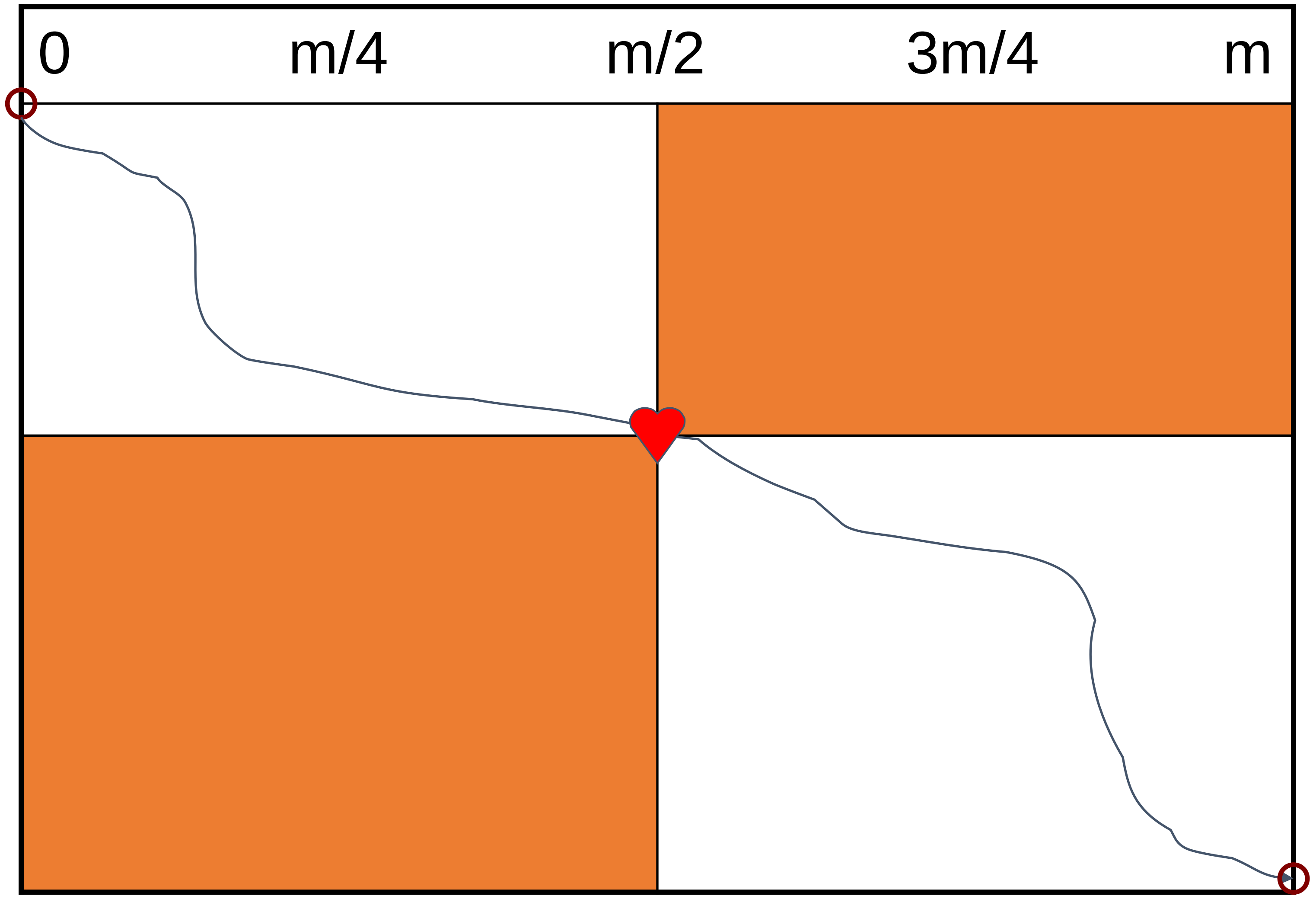


Length = prefix + suffix

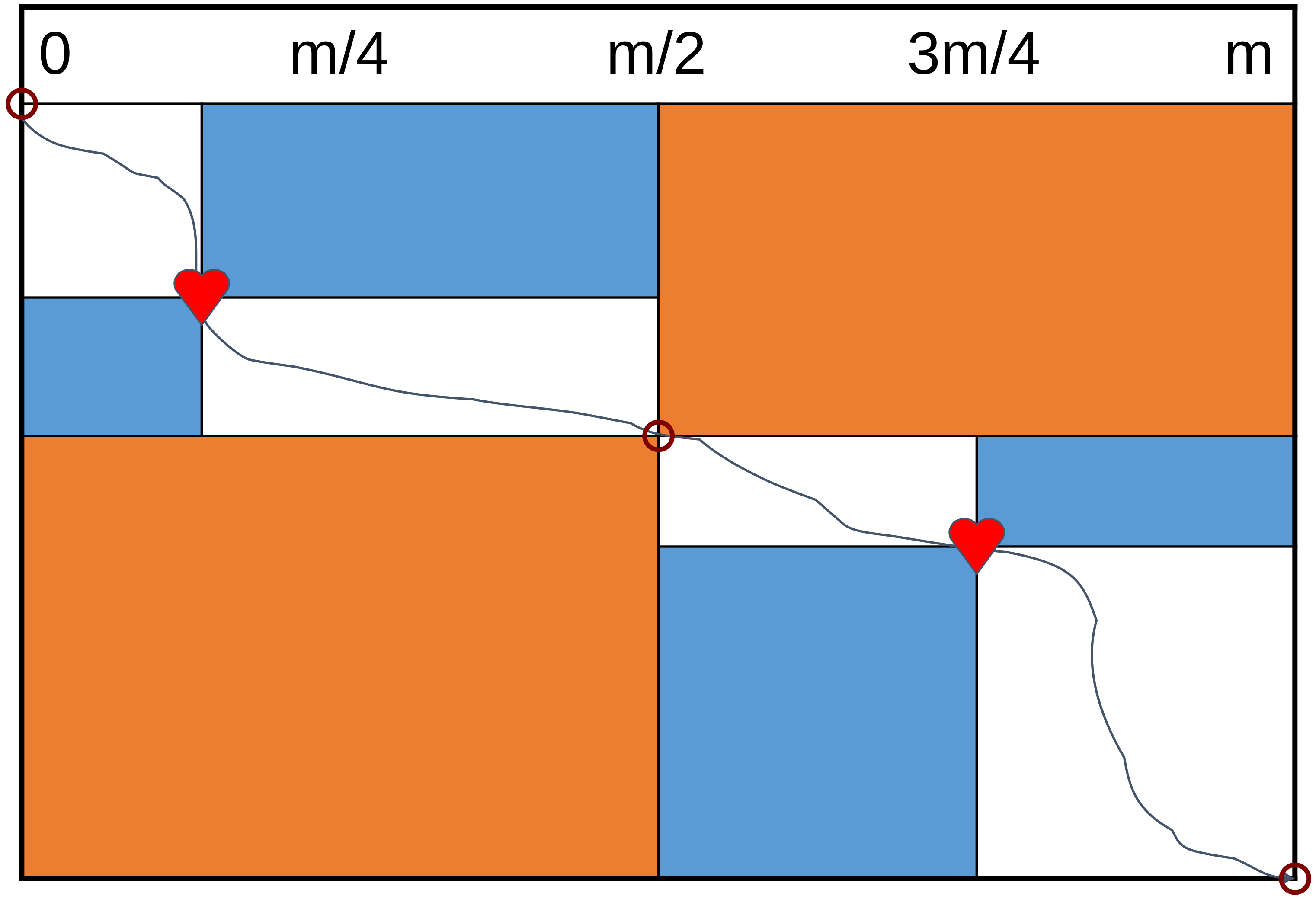
- Add *prefix*(*i*) and *suffix*(*i*) to compute *length(i):*
  - $length(i) = prefix(i) + suffix(i)$
- You now have a middle vertex of the maximum path (*i*,*m*/2) as maximum of *length(i)*



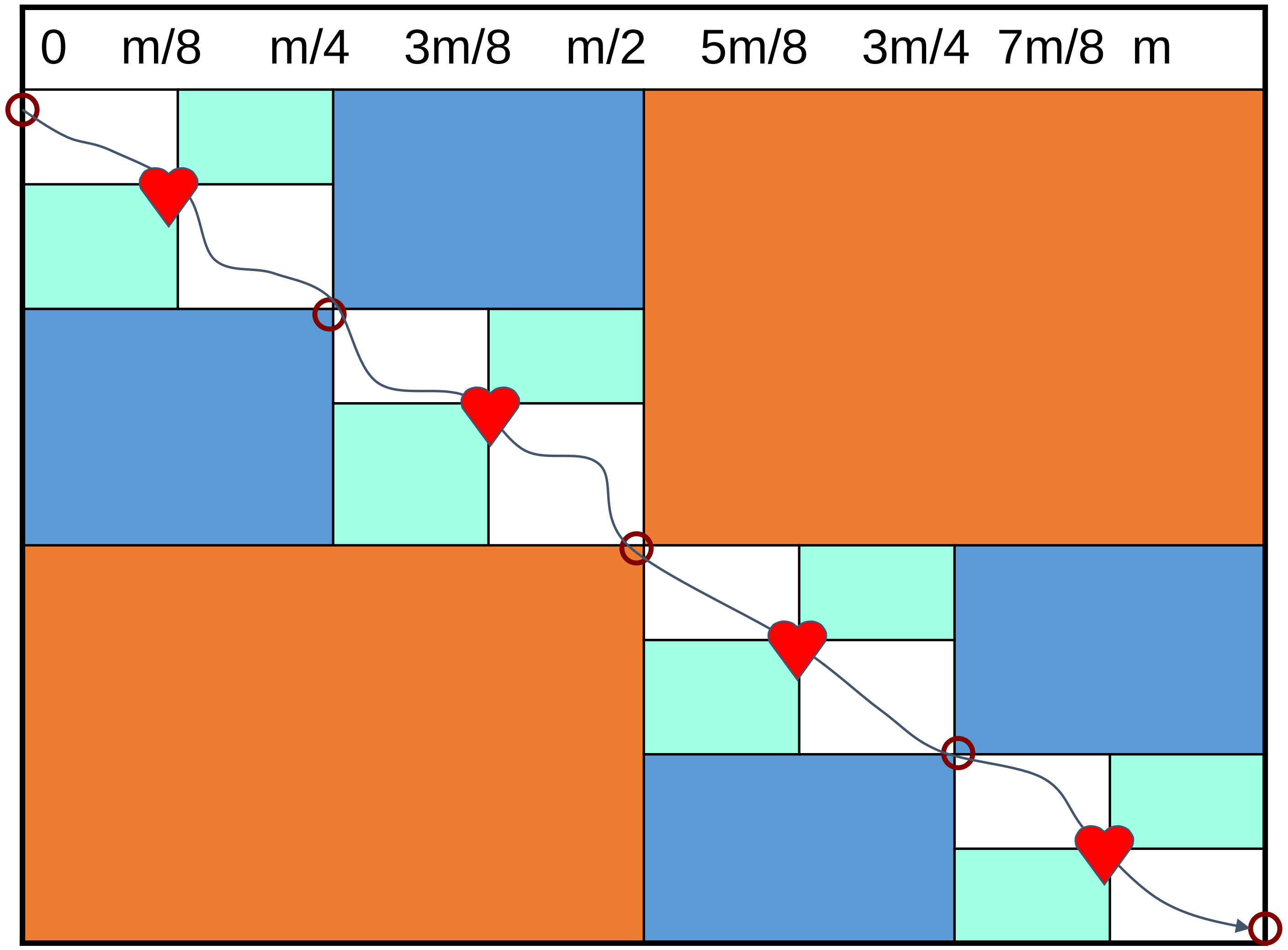
Finding the mid-point



Recursively identify all the mid-points



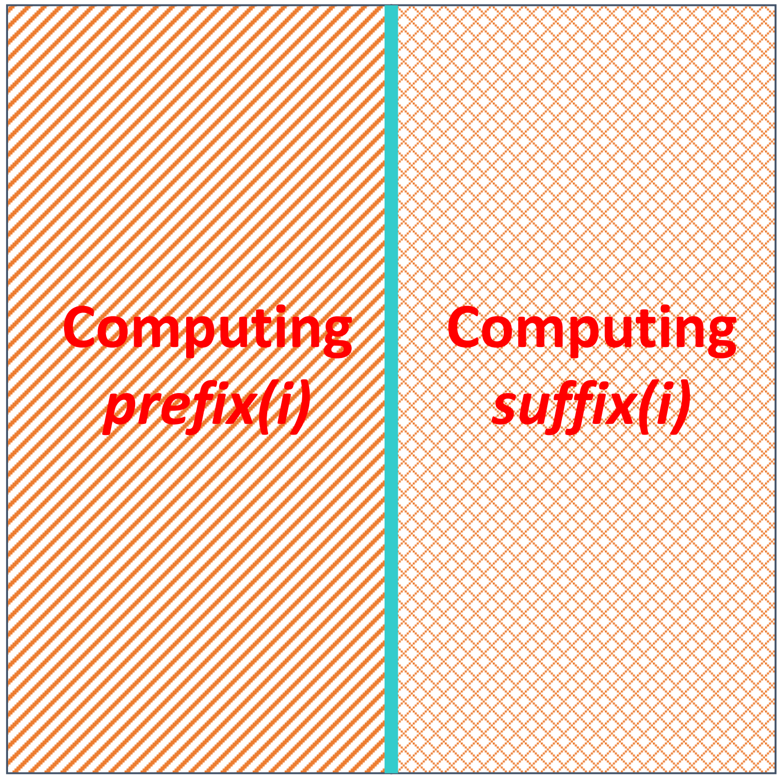
Recursively identify all the mid-points



Time = Area filled

- On first pass, the algorithm covers the entire area

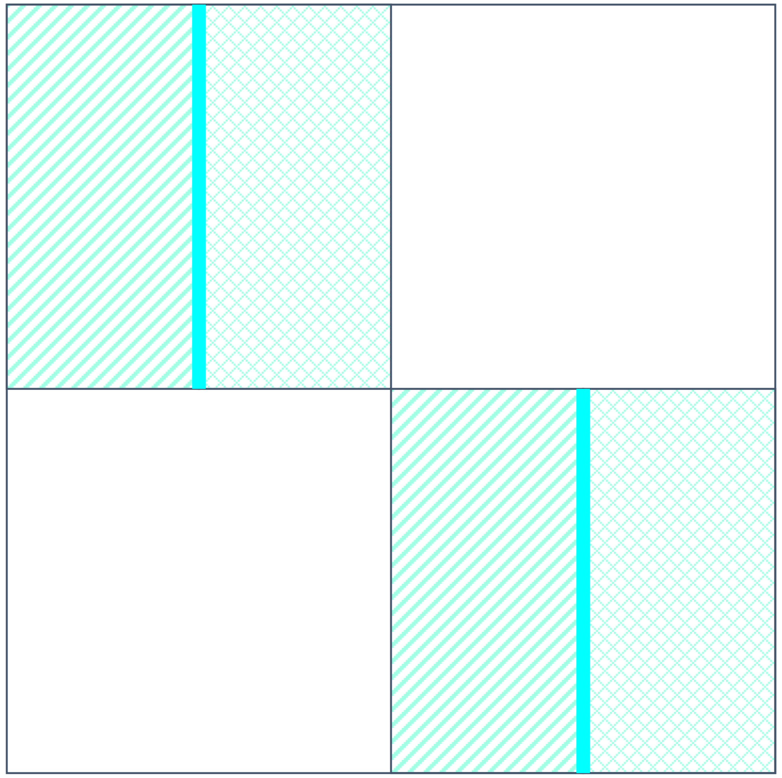
Area $= n \bullet m$



Time = Area filled

- On second pass, the algorithm covers only 1/2 of the area

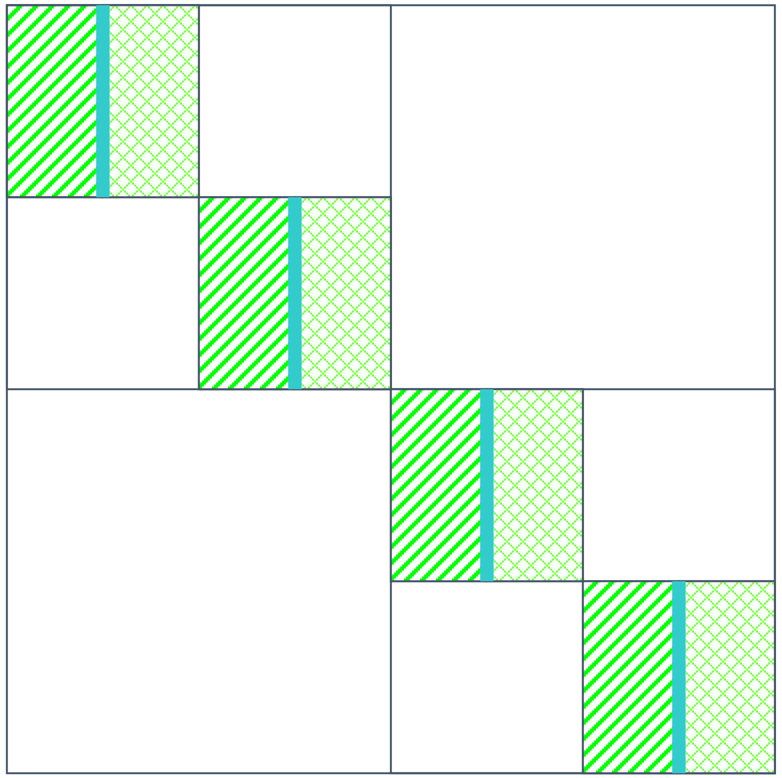
Area/2



Time = Area filled

- On third pass, only 1/4th is covered.

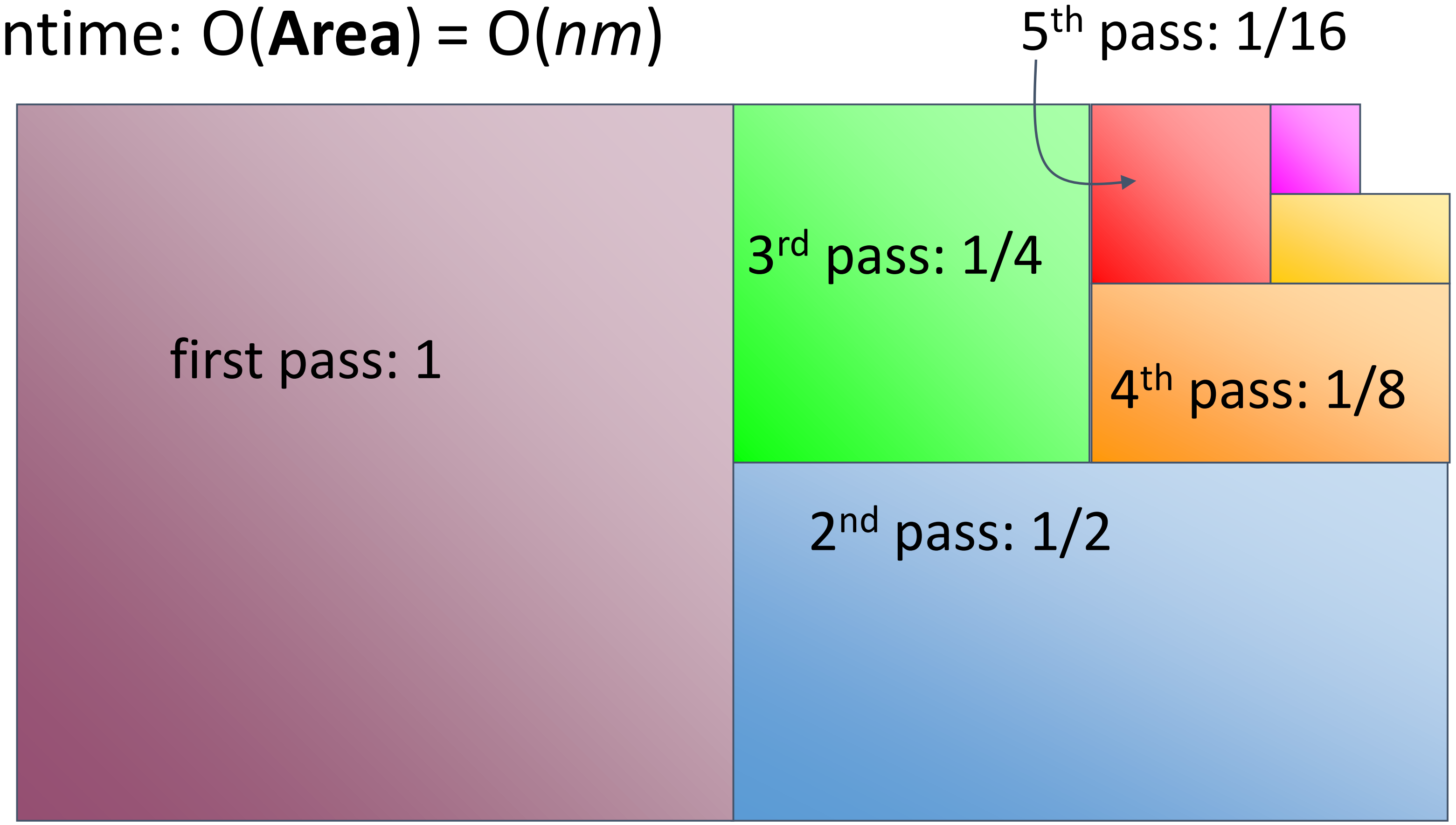
Area/4



Time = Area filled

$1 + ½ + ¼ + … + (½)^k \leq 2$

- Runtime: O(**Area**) = O(*nm*)



More realistic measure of “profit”

- Now we use “score” to represent the “profit”
- We want to maximize the score of the alignment
- Matching identical characters gives positive scores, matching different characters (usually) gives negative scores, introducing gaps gives negative scores

Generalized scoring function

To generalize scoring, consider a (4+1) x(4+1) **scoring matrix** δ.

In the case of an amino acid sequence alignment, the scoring matrix would be a (20+1)x(20+1) size. The addition of 1 is to include the score for comparison of a gap character "-".

This will simplify the algorithm as follows:

$$s_{i,j} = \max \begin{cases} s_{i-1,j-1} + \delta(v_i, w_j) \\ s_{i-1,j} + \delta(v_i, -) \\ s_{i,j-1} + \delta(-, w_j) \end{cases}$$

Scoring matrix

	A	R	N	K
A	5	-2	-1	-1
R	-	7	-1	3
N	-	-	7	0
K	-	-	-	6

AKRANR

KAAANK

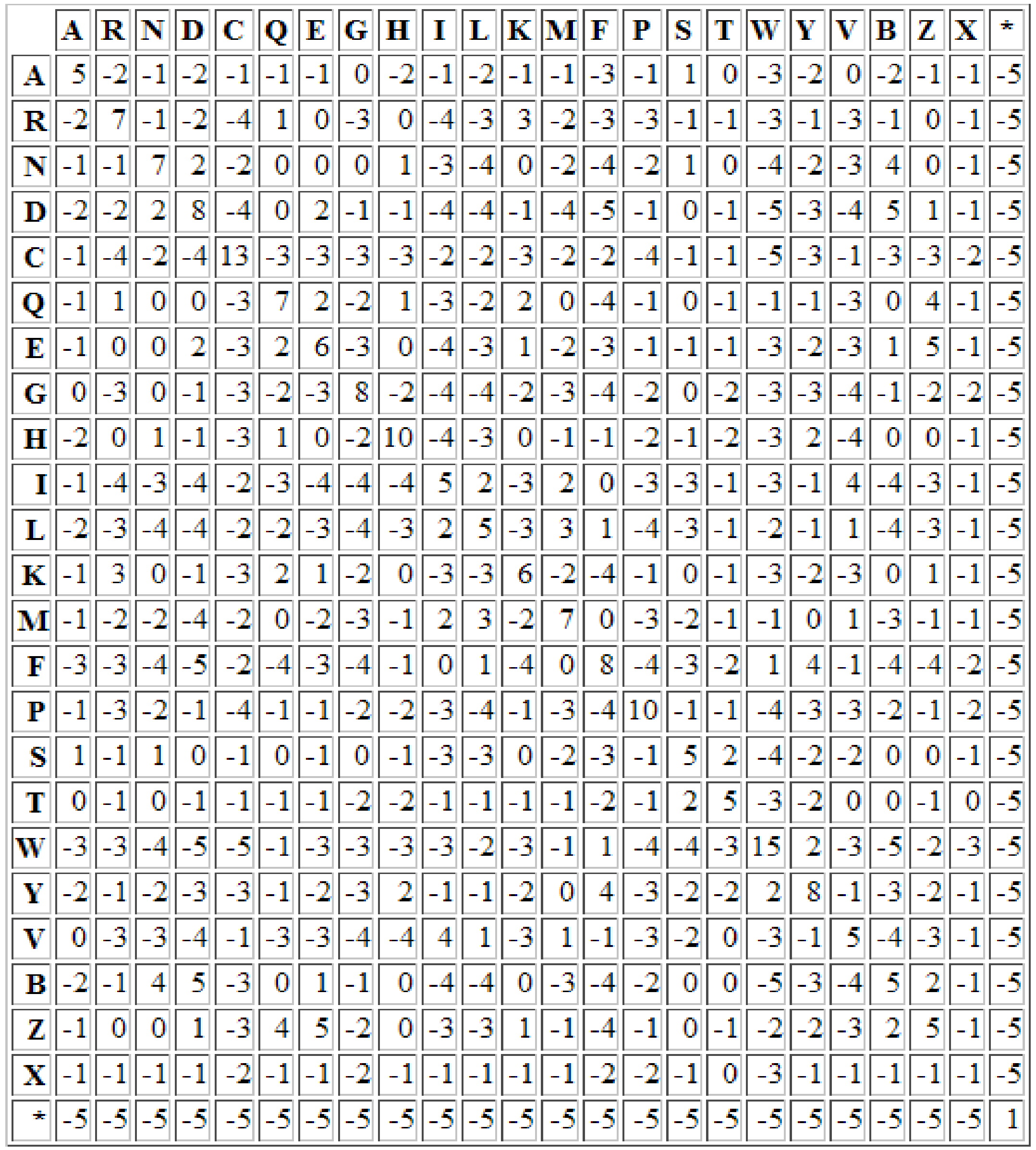
-1 + (-1) + (-2) + 5 + 7 + 3 = 11

- Notice that although R and K are different amino acids, they have a positive score.
- Why? They are both positively charged amino acids→ will not greatly change function of protein.

Scoring matrix cont.

- PAM (**P**oint **A**ccepted **M**utation)
- BLOSUM (**B**lock **S**ubstitution **M**atrix)
- Derived based on known alignments
- Matching characters that tend to present in the same column would have higher score

Scoring matrix example (BLOSUM 50)



	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V	B	Z	X	*
A	5	-2	-1	-2	-1	-1	-1	0	-2	-1	-2	-1	-1	-3	-1	1	0	-3	-2	0	-2	-1	-1	-5
R	-2	7	-1	-2	-4	1	0	-3	0	-4	-3	3	-2	-3	-3	-1	-1	-3	-1	-3	-1	0	-1	-5
N	-1	-1	7	2	-2	0	0	0	1	-3	-4	0	-2	-4	-2	1	0	-4	-2	-3	4	0	-1	-5
D	-2	-2	2	8	-4	0	2	-1	-1	-4	-4	-1	-4	-5	-1	0	-1	-5	-3	-4	5	1	-1	-5
C	-1	-4	-2	-4	13	-3	-3	-3	-3	-2	-2	-3	-2	-2	-4	-1	-1	-5	-3	-1	-3	-3	-2	-5
Q	-1	1	0	0	-3	7	2	-2	1	-3	-2	2	0	-4	-1	0	-1	-1	-1	-3	0	4	-1	-5
E	-1	0	0	2	-3	2	6	-3	0	-4	-3	1	-2	-3	-1	-1	-1	-3	-2	-3	1	5	-1	-5
G	0	-3	0	-1	-3	-2	-3	8	-2	-4	-4	-2	-3	-4	-2	0	-2	-3	-3	-4	-1	-2	-2	-5
H	-2	0	1	-1	-3	1	0	-2	10	-4	-3	0	-1	-1	-2	-1	-2	-3	2	-4	0	0	-1	-5
I	-1	-4	-3	-4	-2	-3	-4	-4	-4	5	2	-3	2	0	-3	-3	-1	-3	-1	4	-4	-3	-1	-5
L	-2	-3	-4	-4	-2	-2	-3	-4	-3	2	5	-3	3	1	-4	-3	-1	-2	-1	1	-4	-3	-1	-5
K	-1	3	0	-1	-3	2	1	-2	0	-3	-3	6	-2	-4	-1	0	-1	-3	-2	-3	0	1	-1	-5
M	-1	-2	-2	-4	-2	0	-2	-3	-1	2	3	-2	7	0	-3	-2	-1	-1	0	1	-3	-1	-1	-5
F	-3	-3	-4	-5	-2	-4	-3	-4	-1	0	1	-4	0	8	-4	-3	-2	1	4	-1	-4	-4	-2	-5
P	-1	-3	-2	-1	-4	-1	-1	-2	-2	-3	-4	-1	-3	-4	10	-1	-1	-4	-3	-3	-2	-1	-2	-5
S	1	-1	1	0	-1	0	-1	0	-1	-3	-3	0	-2	-3	-1	5	2	-4	-2	-2	0	0	-1	-5
T	0	-1	0	-1	-1	-1	-1	-2	-2	-1	-1	-1	-1	-2	-1	2	5	-3	-2	0	0	-1	0	-5
W	-3	-3	-4	-5	-5	-1	-3	-3	-3	-3	-2	-3	-1	1	-4	-4	-3	15	2	-3	-5	-2	-3	-5
Y	-2	-1	-2	-3	-3	-1	-2	-3	2	-1	-1	-2	0	4	-3	-2	-2	2	8	-1	-3	-2	-1	-5
V	0	-3	-3	-4	-1	-3	-3	-4	-4	4	1	-3	1	-1	-3	-2	0	-3	-1	5	-4	-3	-1	-5
B	-2	-1	4	5	-3	0	1	-1	0	-4	-4	0	-3	-4	-2	0	0	-5	-3	-4	5	2	-1	-5
Z	-1	0	0	1	-3	4	5	-2	0	-3	-3	1	-1	-4	-1	0	-1	-2	-2	-3	2	5	-1	-5
X	-1	-1	-1	-1	-2	-1	-1	-2	-1	-1	-1	-1	-1	-2	-2	-1	0	-3	-1	-1	-1	-1	-1	-5
*	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	1

Incorporating such a scoring matrix into our alignment program is trivial !!!

Now the gaps

A fixed penalty σ is given to every indel:

$-\sigma$ for 1 indel,

-2σ for 2 consecutive indels

-3σ for 3 consecutive indels, etc.

```
-C-C-C
 | | |
GCGCGC
```

This is less likely
but scored higher.

```
---CCC
   |*|
GCGCGC
```

This is more likely
but scored lower.

A more realistic mode

- How do you cut out a substring in the middle of a sequence
  - Cut once
  - Cut twice
  - Take out the middle substring
  - Glue the remaining prefix and suffix

- This is also what the nature does!!!

- The major cost of the operation is (**somehow**) independent of the length of the substring being cut out!!!

Affine gap penalty

- *Gaps*- contiguous sequence of spaces in one of the rows

- Score for a gap of length *x* is:

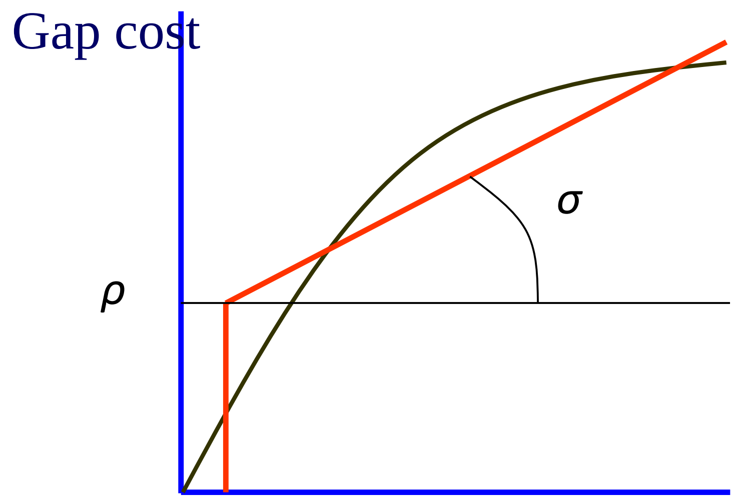
  $-(\rho + \sigma x)$

  where ρ >0 is the penalty for introducing a gap:

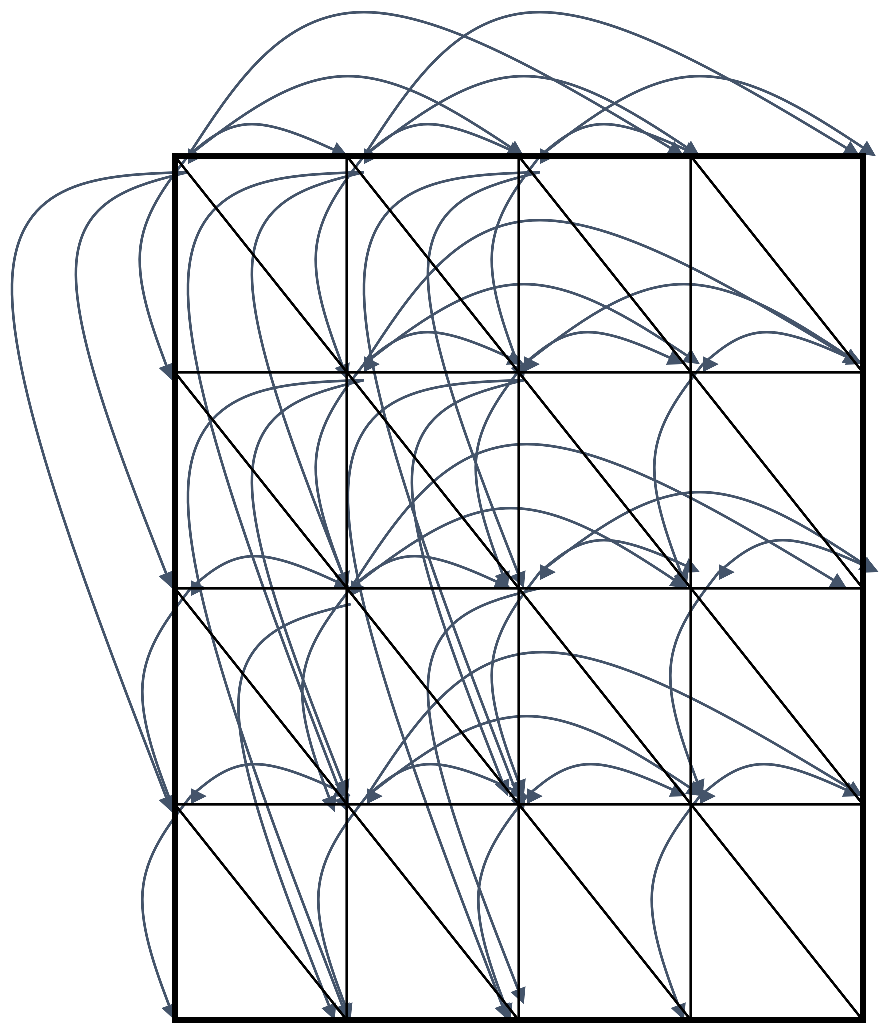
  gap opening penalty

  ρ will be large relative to σ:

  gap extension penalty



Adding affine gap penalty to our algorithm

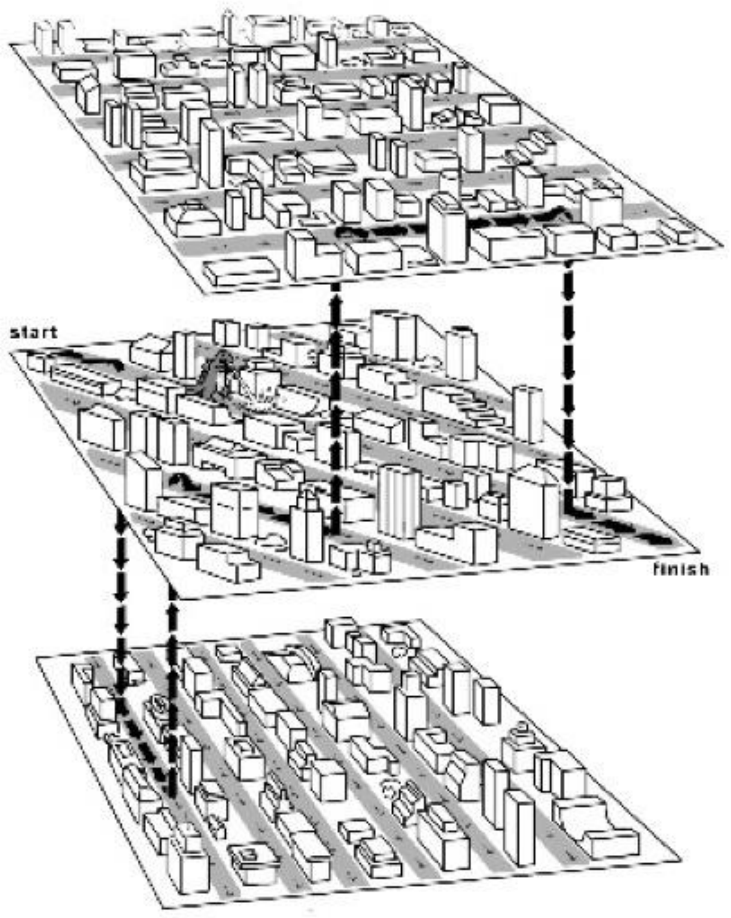


There are many such edges!

Adding them to the graph increases the running time of the alignment algorithm by a factor of ***n*** (where ***n*** is the number of vertices)

So the complexity increases from O(n^2) to O(n^3)

Manhattan in 3 layers

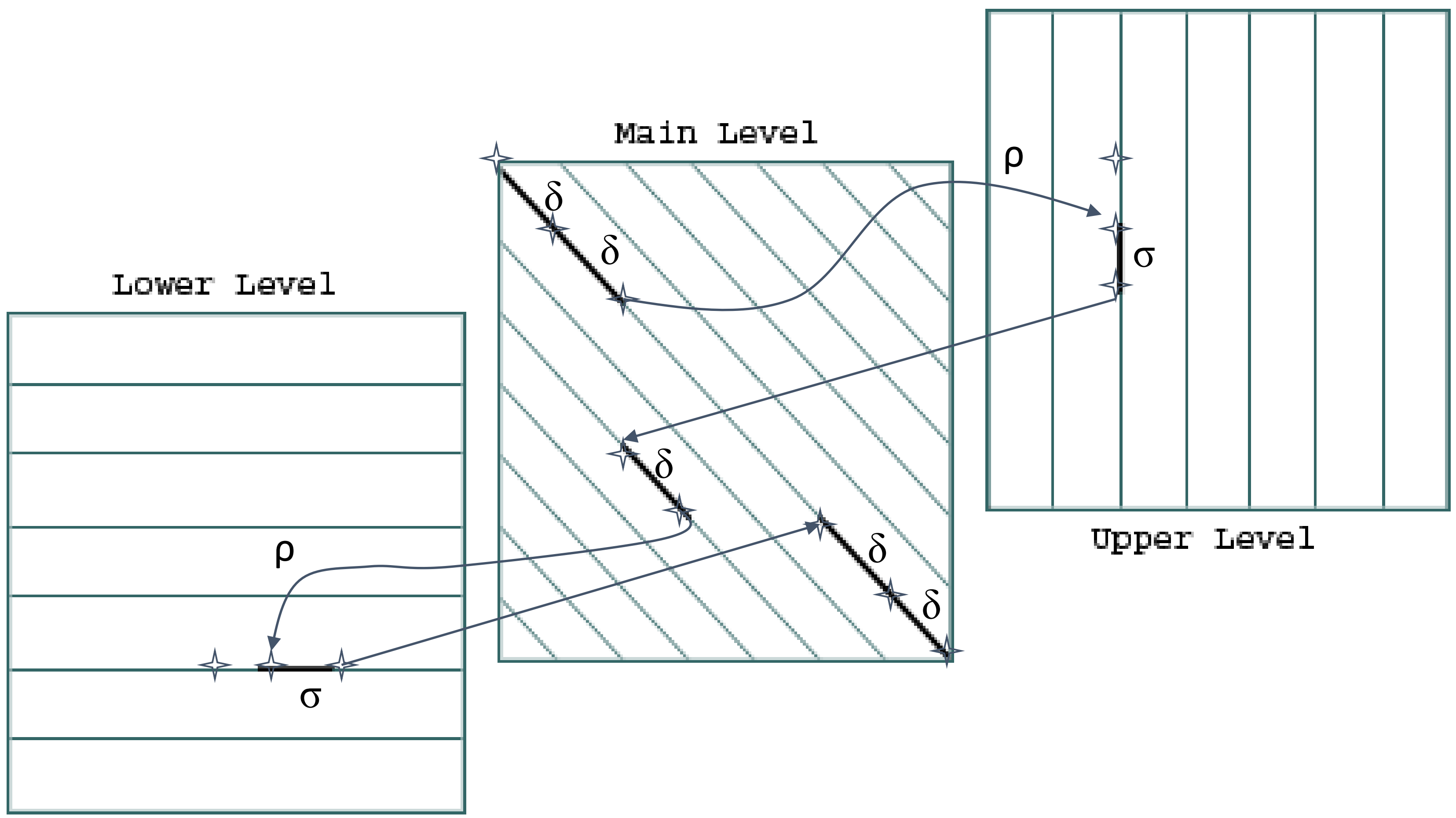


Gaps in w

Matches/Mismatches

Gaps in v

Manhattan in 3 layers



Switching between 3 layers

- Levels:
  - The **main level** is for diagonal edges
  - The **lower level** is for horizontal edges
  - The **upper level** is for vertical edges
- A jumping penalty is assigned to moving from the main level to either the upper level or the lower level ($-\rho - \sigma$)
- There is a gap extension penalty for each continuation on a level other than the main level ($-\sigma$)

Recursion with affine gap penalty

$$\overset{\downarrow}{s}_{i,j} = \max \begin{cases} \overset{\downarrow}{s}_{i-1,j} - \sigma \\ s_{i-1,j} - (\rho + \sigma) \end{cases}$$

Continue Gap in *w* (deletion)

Start Gap in *w* (deletion): from middle

$$\overrightarrow{s}_{i,j} = \max \begin{cases} \overrightarrow{s}_{i,j-1} - \sigma \\ s_{i,j-1} - (\rho + \sigma) \end{cases}$$

Continue Gap in *v* (insertion)

Start Gap in *v* (insertion):from middle

$$s_{i,j} = \max \begin{cases} s_{i-1,j-1} + \delta(v_i, w_j) \\ \overset{\downarrow}{s}_{i,j} \\ \overrightarrow{s}_{i,j} \end{cases}$$

Match or Mismatch

End deletion: from top

End insertion: from bottom