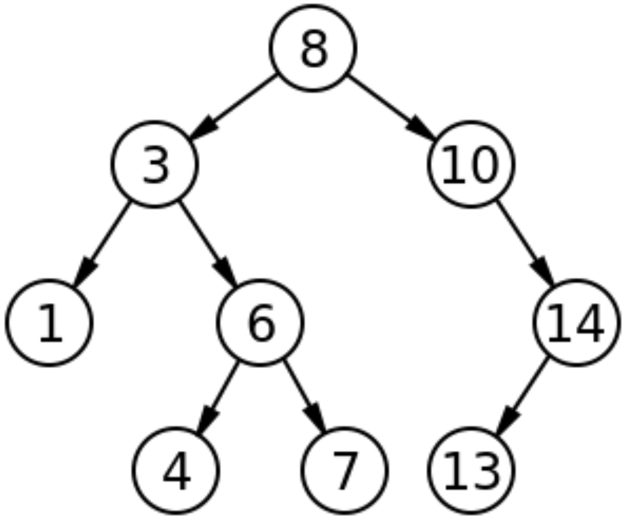
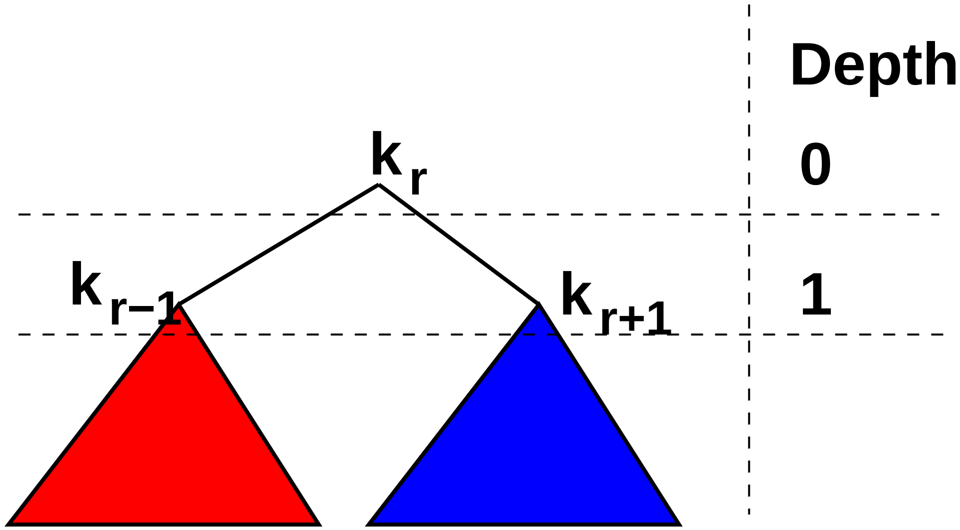
Chapter 6

Dynamic Programming

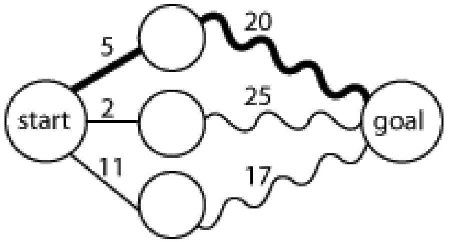




Dynamic Programming

- **Simplex for LP:** Greedy algorithm, makes a **locally optimal** choice.
- For many problems, we need a different approach called **Dynamic Programming**
- Finds solutions for problems with lots of **overlapping sub-problems.** Essentially, we try to solve each sub-problem **only once**.
- **Optimal substructure:** optimal solutions of **subproblems** can be used to find the optimal solutions of the **overall problem**.

Example: Finding the shortest path in a graph.



Dynamic Programming

Typically, a dynamic programming solution is constructed using a **series of steps:**

1. Characterise the **structure** of an optimal solution.

2. **Recursively** define the value of an **optimal solution**.

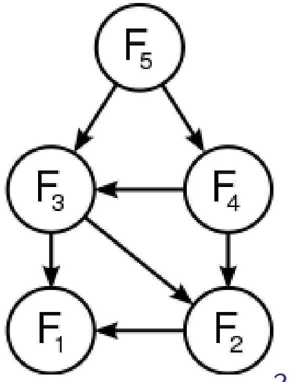
3. Compute the value of an optimal solution in a **bottom-up** ($\leadsto$ iteration) or **top-down** ($\leadsto$ recursion) fashion. That is, build it from the results of **smaller solutions** either **iteratively** from the bottom or **recursively** from the top.

A Simple Example: Fibonacci numbers

Fibonacci sequence: The n-th number is the sum of the previous two. This can be implemented using a simple recursive algorithm:

```
function FIBONACCI($n$)
    if $n = 0$ then
        return 0
    if $n = 1$ then
        return 1
    return FIBONACCI($n - 1$) + FIBONACCI($n - 2$)
```

Problem: Overlapping sub-problems: Computing FIBONACCI($n - 1$) overlaps FIBONACCI($n - 2$)
⇝ exponential time complexity!



A Simple Example (2)

Define **map object** m, maps each instance of FIBONACCI that has already been calculated to its result.

Modified recursion requires only $O(n)$ time:

```
**var** $m$; $m[0] = 0$;   $m[1] = 1$
**function** FIBONACCI($n$)
    **if** $m$ does not contain key $n$
        $m[n] =$ FIBONACCI($n-1$) + FIBONACCI($n-2$)
    **return** $m[n]$
```

Or define array f and use **iteration:** $f[0] = 0$, $f[1] = 1$.

```
FIBONACCI($n$)
   for $i = 2$ upto $n$ step $1$ do
       $f[i] = f[i-1] + f[i-2]$
    **return** $f[n]$
```

Another Example: Optimal Binary Search Trees

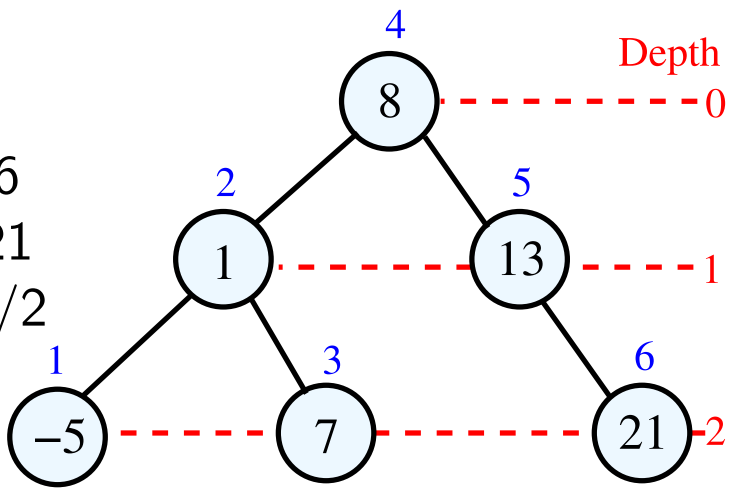
BST: Tree where the key values are stored in the nodes, and the keys are ordered lexicographically.

For each internal node all keys in the left subtree are less than the keys in the node, and all the keys in the right subtree are greater.

Knowing the probabilities of searching each one of the keys makes it easy to compute the expected cost of accessing the tree.

An **OBST** is a BST with *minimal expected search costs*.

Index	1	2	3	4	5	6
Keys	-5	1	7	8	13	21
Probabilities	$1/8$	$1/32$	$1/32$	$1/16$	$1/4$	$1/2$



OBST

- **Keys** $k_1, \ldots, k_n$ in lexicographical order,

- **Probabilities** of acessing keys $p_1, \ldots, p_n$.

- **Depth** $D_T(k_m)$ of node k_m in tree T. $D_T(\text{root}) = 0$

- T^{ij}: tree constructed from keys $k_i, \ldots, k_j$

- **Costs:** number of comparisons done in a search.

- **Expected costs:** expected number of comparisons done during search in tree, given the acess probabilities p_i

OBST: Expected costs

Definiton of expected costs of tree constructed from keys $k_i, \ldots, k_j$:

$$C_{i,j} := E[cost(T^{ij})]$$

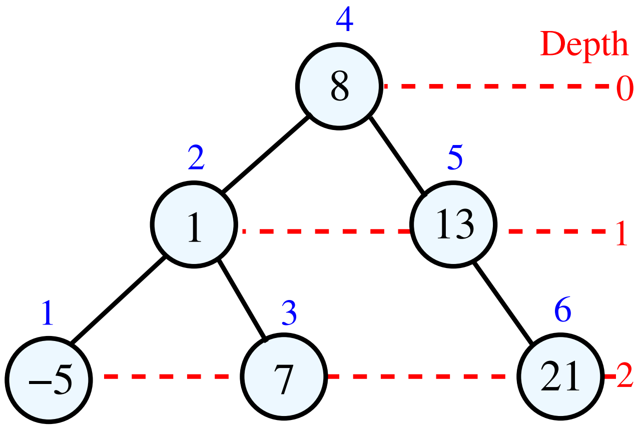
$$= \sum_{\text{all keys in } T^{ij}} \text{prob. of key} \times (\text{depth of key} \overbrace{+1}^{\text{one comparison for root}})$$

$$= \sum_{m=i}^{j} p_m(D_T(k_m) + 1)$$

Index	1	2	3	4	5	6
Keys	-5	1	7	8	13	21
Probabilities	1/8	1/32	1/32	1/16	1/4	1/2

$C_{1,6} = 1 \cdot 1/16 + 2 \cdot (1/32 + 1/4) + 3 \cdot (1/8 + 1/32 + 1/2)$

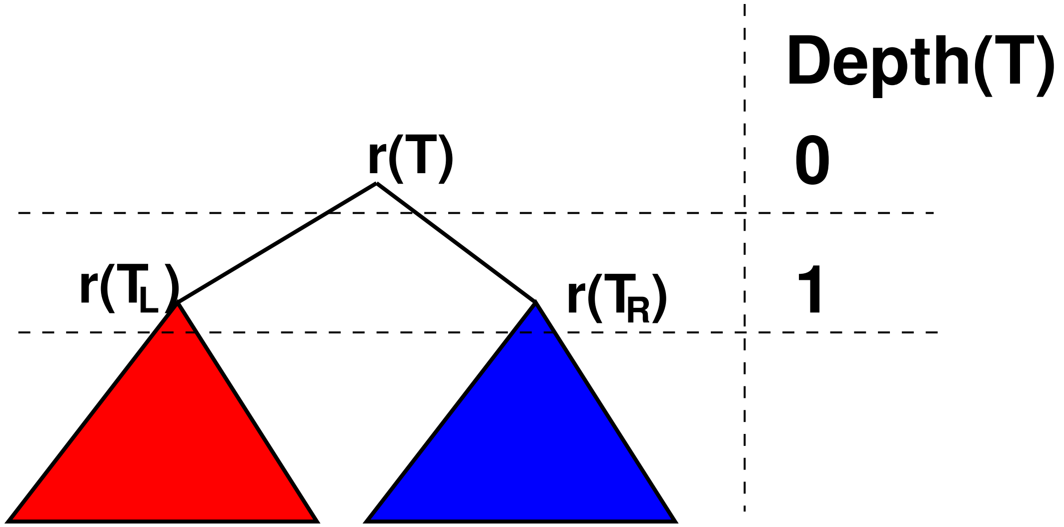
$= 85/32$



OBST

- **Key observation:** each subtree of an optimal tree is itself optimal (replacing a subtree with a better one lowers the costs of entire tree)

- Consider tree T^{ij} with root node $r(T) = k_r$.



Expected costs of tree $T = T^{ij}$

$$
\begin{aligned}
C_{i,j} &= \sum_{m=i}^{j} p_m(D_T(k_m) + 1) \\
&= \sum_{m=i}^{r-1} p_m(D_T(k_m) + 1) + p_r + \sum_{m=r+1}^{j} p_m(D_T(k_m) + 1) \\
&= \underbrace{\sum_{m=i}^{r-1} p_m((D_{T_L}(k_m) + 1) + 1)}_{\text{C(left subtree)}} + \underbrace{p_r}_{\text{root}} + \underbrace{\sum_{m=r+1}^{j} p_m((D_{T_R}(k_m) + 1) + 1)}_{\text{C(right subtree)}} \\
&= C(T_L) + \sum_{m=i}^{r-1} p_m + p_r + C(T_R) + \sum_{m=r+1}^{j} p_m \\
&= C(T_L) + C(T_R) + \sum_{m=i}^{j} p_m
\end{aligned}
$$

OBST: algorithm

Recursive algorithm:

- consider every node as being the root
- split rest of the keys into left and right subtrees and recursively calculate their costs.

$$
\begin{aligned}
C_{i,i} &= p_i \\
C_{i,j} &= 0 \, \forall \, j < i \quad \text{(tree with no nodes)} \\
C_{i,j} &= \sum_{m=i}^{j} p_m + \min_{i \le r \le j} \left[C_{i,r-1} + C_{r+1,j} \right]
\end{aligned}
$$

Use **memoization** to avoid solving the same problem over and over.
Or use **iterative** algorithm.

DP for an OBST

- Precompute $P_{ij} = \sum_{m=i}^{j} p_m$.
- Fill C-matrix by diagonals (start with main diagonal, move up-right)
- Store “winning” root index in matrix R

$$(\mathbf{C})_{ij} = \begin{array}{c|c|c|c|c|c|c|}
 & 1 & 2 & 3 & 4 & 5 & 6 \\ \hline
1 & \frac{1}{8} & \frac{3}{16} & \frac{9}{32} & \frac{15}{32} & \frac{31}{32} & \frac{63}{32} \\ \hline
2 & 0 & \frac{1}{32} & \frac{3}{32} & \frac{7}{32} & \frac{19}{32} & \frac{47}{32} \\ \hline
3 & & 0 & \frac{1}{32} & \frac{1}{8} & \frac{15}{32} & \frac{21}{16} \\ \hline
4 & & & 0 & \frac{1}{16} & \frac{3}{8} & \frac{19}{16} \\ \hline
5 & & & & 0 & \frac{1}{4} & 1 \\ \hline
6 & & & & & 0 & \frac{1}{2} \\ \hline
\end{array} = \frac{1}{32}\begin{pmatrix}
4 & 6 & 9 & 15 & 31 & 63 \\
 & 1 & 3 & 7 & 19 & 47 \\
 & & 1 & 4 & 15 & 42 \\
 & & & 2 & 12 & 38 \\
 & & & & 8 & 32 \\
 & & & & & 16
\end{pmatrix}$$

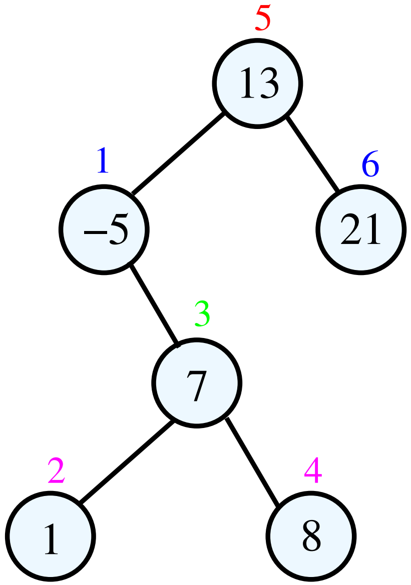
- Find tree by backtracking: start in upper right corner $R_{1,n}$
  $\rightsquigarrow$ **root of full tree**, say root $= k$.
  Right subtree: proceed with $R(k+1,n)$
  $\rightsquigarrow$ **root of right subtree** $T_{k+1,n}$, say $R(k+1,n) = r$.
  Draw edge $k \to r$.
  Left subtree: $R(1,k-1) = l \rightsquigarrow$ **root of left subtree**, edge $k \to l$.
  Recurse.

Computations

In our case:

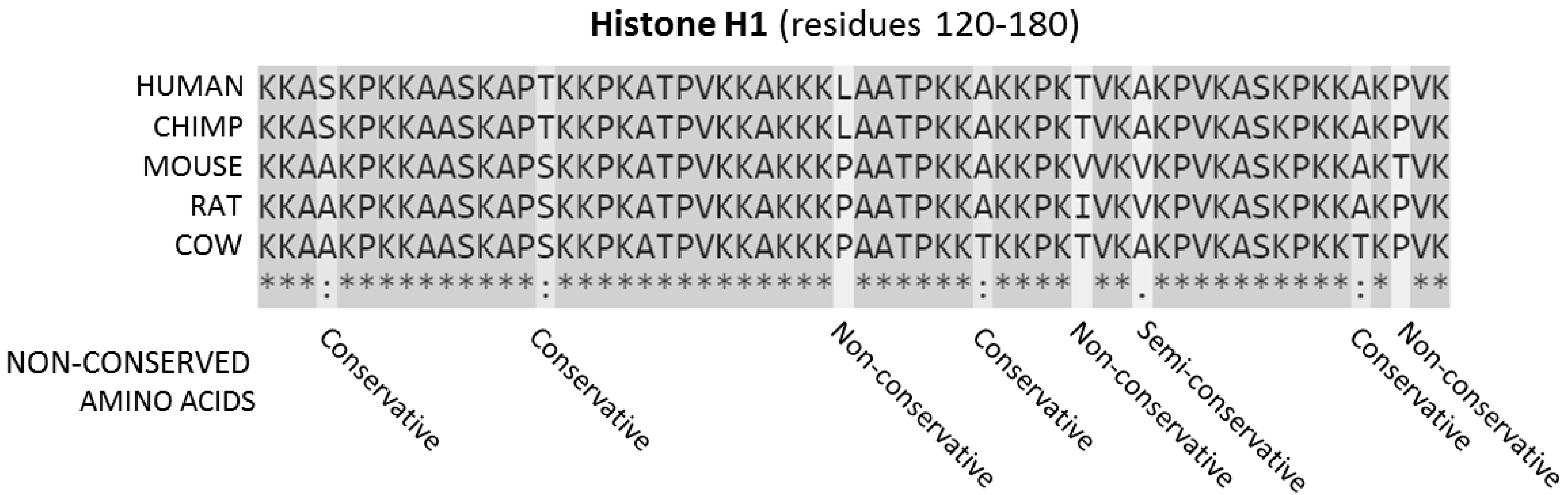
Index	1	2	3	4	5	6
Keys	-5	1	7	8	13	21
Probabilities	1/8	1/32	1/32	1/16	1/4	1/2

$$R = \begin{pmatrix} 1 & 1 & 1 & 1 & 5 & 5 \\ & 2 & 2 & 3 & 5 & 6 \\ & & 3 & 4 & 5 & 6 \\ & & & 4 & 5 & 6 \\ & & & & 5 & 6 \\ & & & & & 6 \end{pmatrix}$$



$$\begin{aligned} E[cost] &= 1 \cdot 1/4 + 2 \cdot (1/2 + 1/8) + 3 \cdot (1/32) + 4 \cdot (1/16 + 1/32) \\ &= 1/32[8 + 2(16+4) + 3 + 4(2+1)] = 63/32. \end{aligned}$$

DP for Aligning Biological Sequences



Histone H1 (residues 120-180)

```
HUMAN KKASKPKKAASKAPTKKPKATPVKKAKKKLAATPKKAKKPKTVKAKPVKASKPKKAKPVK
CHIMP KKASKPKKAASKAPTKKPKATPVKKAKKKLAATPKKAKKPKTVKAKPVKASKPKKAKPVK
MOUSE KKAAKPKKAASKAPSKKPKATPVKKAKKKPAATPKKAKKPKVVKVKPVKASKPKKAKTVK
  RAT KKAAKPKKAASKAPSKKPKATPVKKAKKKPAATPKKAKKPKIVKVKPVKASKPKKAKPVK
  COW KKAAKPKKAASKAPSKKPKATPVKKAKKKPAATPKKTKKPKTVKAKPVKASKPKKTKPVK
      ***:**********:************** ******:**** **.**********:* **
```

By

Thomas Shafee - Own work, CC BY 4.0, https://commons.wikimedia.org/w/index.php?curid=37188728

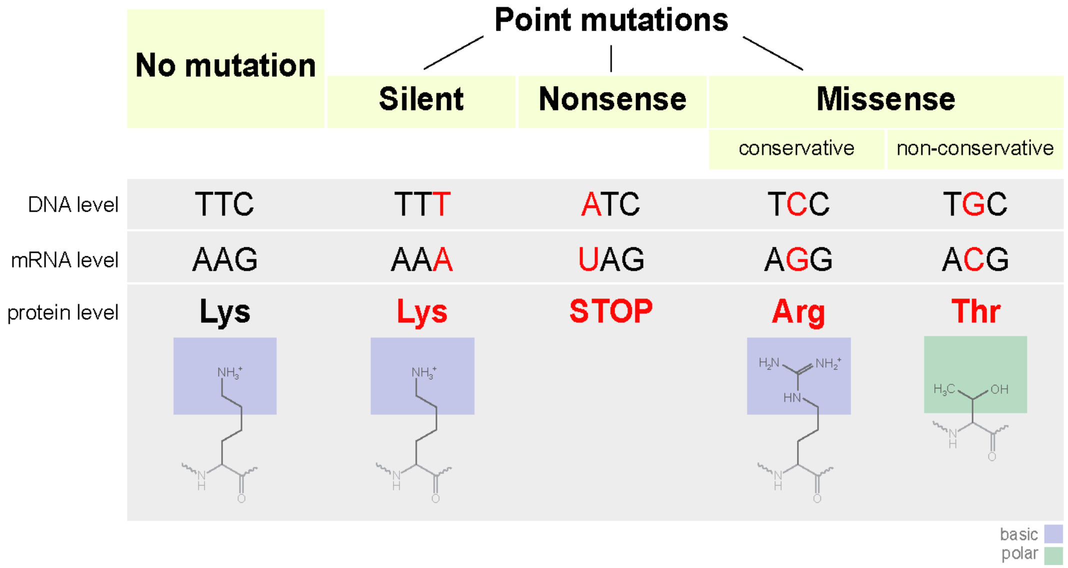
Mutations

- **Mutation**: Heritable change in the DNA sequence. Occur due to exposure to **ultra violet radiation** or other **environmental conditions**.

- **Two levels** at which a mutation can take place:
  - **Point mutation:** within a single gene.
    - **substitution** (change of one nucleotide),
    - **insertion** (addition of nucleotides),
    - **deletion**.
  - **Chromosomal mutation:** whole segments interchange, either on the same chromosome, or on different ones.

Point Mutations

- May arise from **spontaneous mutations** during **DNA replication.**
- The rate of mutation increased by **mutagens** (physical or chemical agent that changes the genetic material).
- Mutagens: Physical (UV-, X-rays or heat), or chemical (molecules misplace base pairs / disrupt helical shape of DNA).



	No mutation	Point mutations			
		Silent	Nonsense	Missense	
				conservative	non-conservative
DNA level	TTC	TTT	ATC	TCC	TGC
mRNA level	AAG	AAA	UAG	AGG	ACG
protein level	**Lys**	**Lys**	**STOP**	**Arg**	**Thr**

Wikipedia. By Jonsta247 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=12481467

Importance of Mutations

- Mutations are responsible for **inherited disorders & diseases.**
  Sickle-cell anemia caused by missense point mutation in **hemoglobin**
  (in blood cells, responsible for oxygen transport.)
  Hydrophilic **glutamic acid** replaced with hydrophobic **valine.**
  ⇝ deformed red blood cells.

  Sequence for Normal Hemoglobin: 6th codon: **adenine** (A)

AUG	GUG	CAC	CUG	ACU	CCU	GAG	GAG	AAG	UCU	GCC	GUU	ACU
START	Val	His	Leu	Thr	Pro	Glu	Glu	Lys	Ser	Ala	Val	Thr

  Sickle Cell Hemoglobin: ⇝ **thymine** (DNA), **uracil** (RNA)

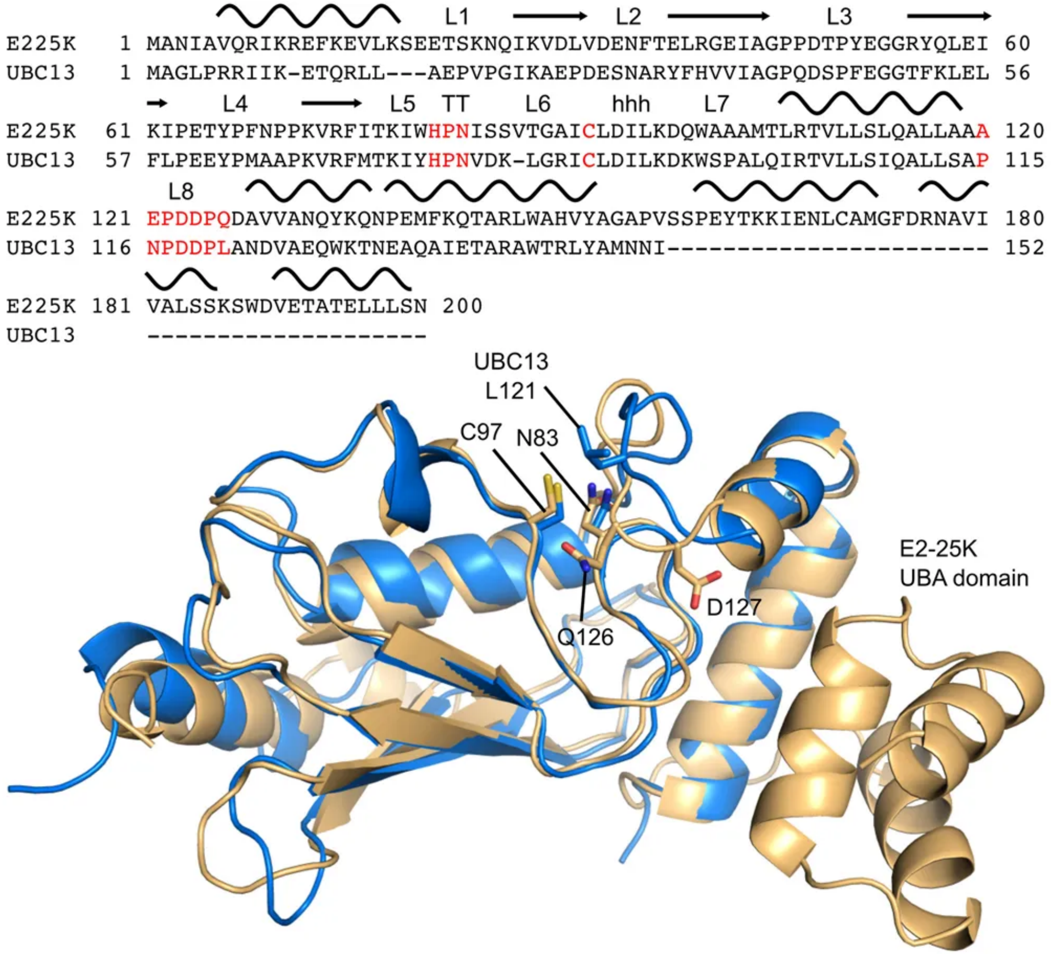
AUG	GUG	CAC	CUG	ACU	CCU	GUG	GAG	AAG	UCU	GCC	GUU	ACU
START	Val	His	Leu	Thr	Pro	Val	Glu	Lys	Ser	Ala	Val	Thr

- Mutations are the source of **phenotypic variation**
  ⇒ **new species** and **adaption** to environmental conditions.

Sequence Comparison: Motivation

Basic idea: **similar sequences** ⇝ **similar proteins.**

Protein folding: 30 % sequence identity ⇒ structures similar.



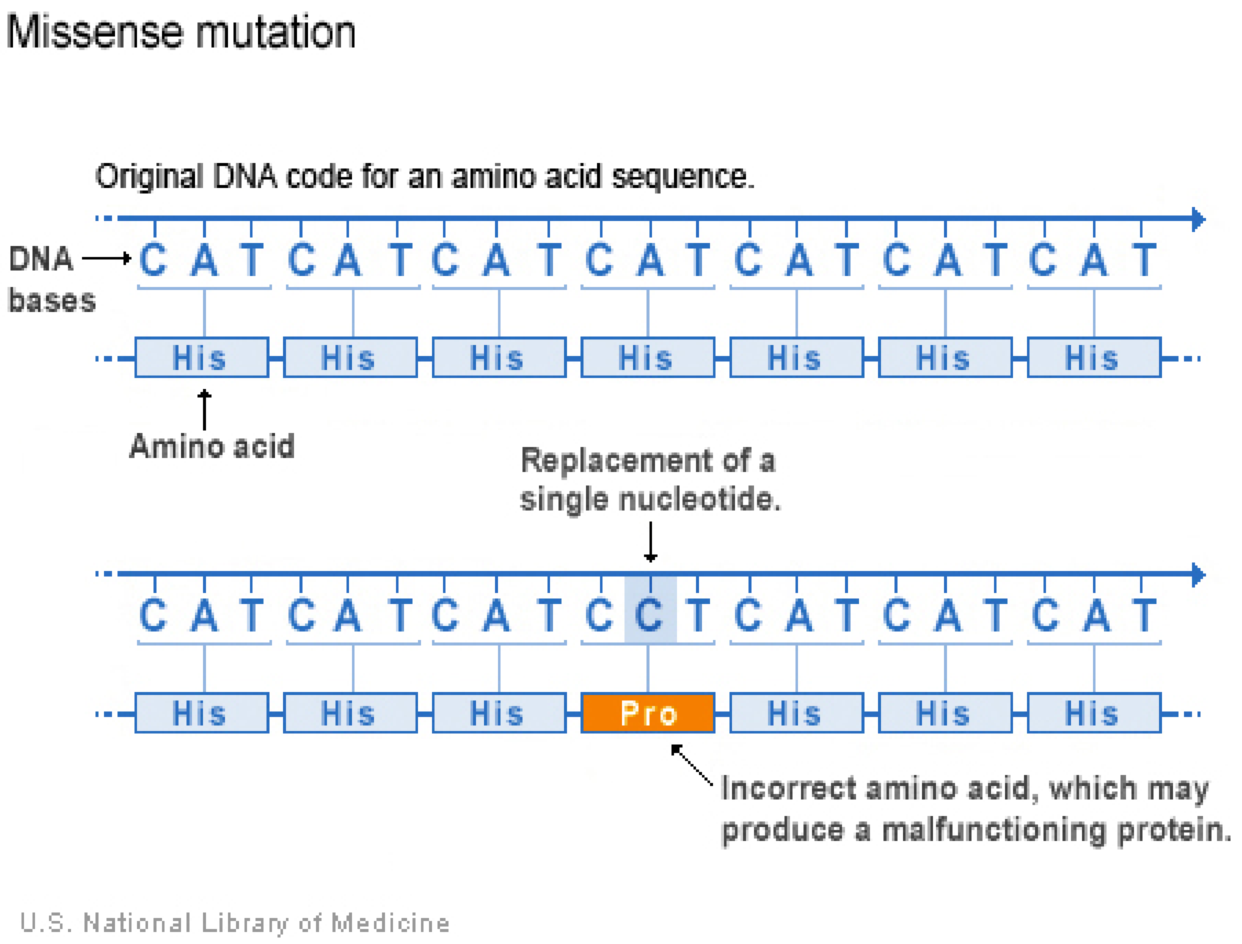
```
                                 L1           L2             L3
E225K    1 MANIAVQRIKREFKEVLKSEETSKNQIKVDLVDENFTELRGEIAGPPDTPYEGGRYQLEI 60
UBC13    1 MAGLPRRIIK-ETQRLL---AEPVPGIKAEPDESNARYFHVVIAGPQDSPFEGGTFKLEL 56
                L4         L5 TT     L6     hhh    L7
E225K   61 KIPETYPFNPPKVRFITKIWHPNISSVTGAICLDILKDQWAAAMTLRTVLLSLQALLAAA 120
UBC13   57 FLPEEYPMAAPKVRFMTKIYHPNVDK-LGRICLDILKDKWSPALQIRTVLLSIQALLSAP 115
             L8
E225K  121 EPDDPQDAVVANQYKQNPEMFKQTARLWAHVYAGAPVSSPEYTKKIENLCAMGFDRNAVI 180
UBC13  116 NPDDPLANDVAEQWKTNEAQAIETARAWTRLYAMNNI----------------------- 152

E225K  181 VALSSKSWDVETATELLLSN 200
UBC13      --------------------
```

Rout et al., Scientific Reports, vol 8, no 7002 (2018)

Comparing sequences

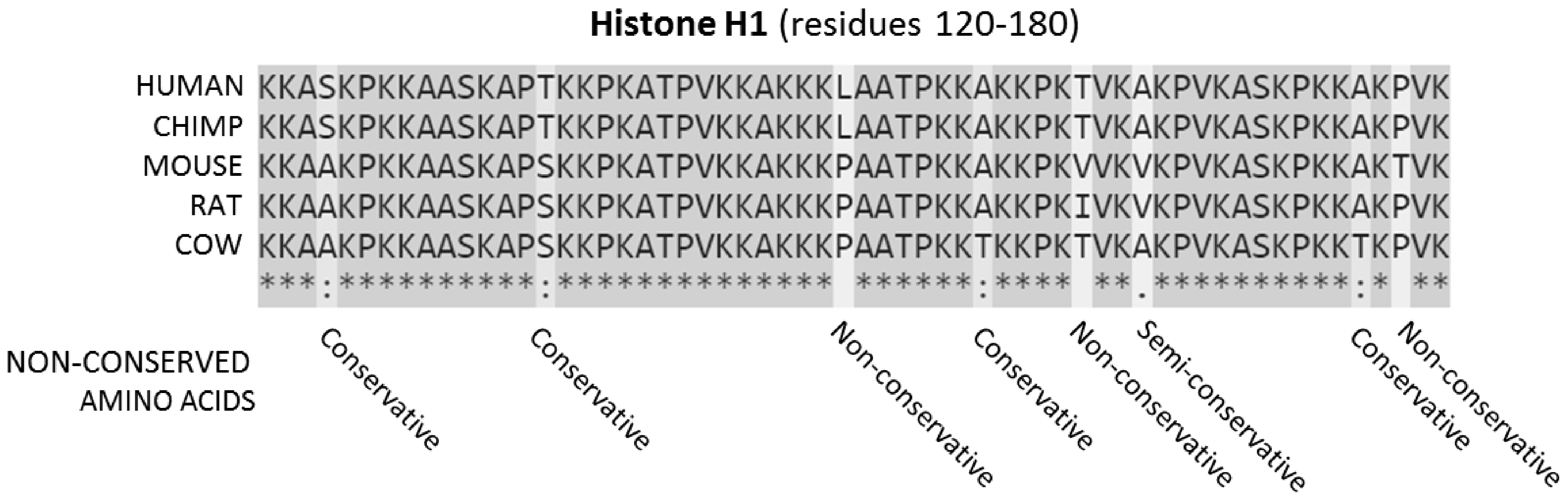
Theory: during evolution **mutations** occurred, creating differences between families of contemporary species.


Missense mutation
Original DNA code for an amino acid sequence.
DNA bases
C A T C A T C A T C A T C A T C A T C A T
His His His His His His His
Amino acid
Replacement of a single nucleotide.
C A T C A T C A T C C T C A T C A T C A T
His His His Pro His His His
Incorrect amino acid, which may produce a malfunctioning protein.
U.S. National Library of Medicine

https://commons.wikimedia.org/w/index.php?curid=25399199

Comparing sequences

Comparing two sequences: looking for **evidence** that they have **diverged from a common ancestor** by a **mutation process**.



```
       Histone H1 (residues 120-180)
HUMAN  KKASKPKKAASKAPTKKPKATPVKKAKKKLAATPKKAKKPKTVKAKPVKASKPKKAKPVK
CHIMP  KKASKPKKAASKAPTKKPKATPVKKAKKKLAATPKKAKKPKTVKAKPVKASKPKKAKPVK
MOUSE  KKAAKPKKAASKAPSKKPKATPVKKAKKKPAATPKKAKKPKVVKVKPVKASKPKKAKTVK
RAT    KKAAKPKKAASKAPSKKPKATPVKKAKKKPAATPKKAKKPKIVKVKPVKASKPKKAKPVK
COW    KKAAKPKKAASKAPSKKPKATPVKKAKKKPAATPKKTKKPKTVKAKPVKASKPKKTKPVK
       ***:**********:************** ******:**** **.**********:* **
```

By

Thomas Shafee - Own work, CC BY 4.0, https://commons.wikimedia.org/w/index.php?curid=37188728

Sequence Alignment

Informal definition:
Alignment of sequences $x = x_1 \ldots x_n$ and $y = y_1 \ldots y_m$:
(i) **insert spaces**,
(ii) place resulting sequences **one above the other** so that every character or space has a counterpart.

Example: `ACBCDDDB` and `CADBDAD`. Possible alignments:

```
A C - - B C D D D B
  |     |   |   |
- C A D B - D A D -

- A C B C D D D B
  |   |     |
C A D B D A D - -
```

Optimal Alignment

Given: two sequences x and y over alphabet $\mathcal{A}$.

$\mathcal{A} = \{\texttt{A},\texttt{G},\texttt{C},\texttt{T}\}$ (DNA)
$\mathcal{A} = \{\texttt{A},\texttt{R},\texttt{N},\texttt{D},\texttt{C},\texttt{Q},\texttt{E},\texttt{G},\texttt{H},\texttt{I},\texttt{L},\texttt{K},\texttt{M},\texttt{F},\texttt{P},\texttt{S},\texttt{T},\texttt{W},\texttt{Y},\texttt{V}\}$ (proteins)

Formalizing **optimality of an alignment**: define

- the costs for **substituting** a letter by another letter ⇒ **substitution matrix**;
- the costs for **insertion** ⇒ **gap penalties**.

The Scoring Model

- **Idea:** assign a score to each alignment, choose best one.

- **Additive** scoring scheme: Total score = sum of all scores for pairs of letters + costs for gaps.
  Implicit assumption:
  Mutations at different sites have occurred **independently**.
  (In most cases) reasonable for DNA and protein sequences.

- **All** common algorithms use **additive scoring schemes.**

- Modeling dependencies is possible, but at the price of significant computational complexities.

Substitution Matrices

- **Expectation:**
  Identities in real alignments are more likely than by chance.
- Derive score for aligned pairs from a **probabilistic model.**
- **Score:** relative likelihood that two sequences are evolutionary related as opposed to being unrelated
  $\rightsquigarrow$ **score = ratio of probabilities.**
- **First assumption:** Ungapped alignment, $n = m$.
- R: **Random model:**
  Letter a occurs **independently** with some frequency q_a
  $\Rightarrow$ Pr(two sequences) $=$ product of probabilities for each letter:

$$P(x, y|R) = \prod_i q_{x_i} \prod_i q_{y_i}.$$

Substitution Matrices

- M (**match**): aligned pairs occur with **joint probability**

$$P(x,y|M) = \prod_i p_{x_i y_i}$$

- Ratio $\rightsquigarrow$ **“odds ratio”**:

$$\frac{P(x,y|M)}{P(x,y|R)} = \prod_i \frac{p_{x_i y_i}}{q_{x_i} q_{y_i}}$$

- To arrive at an **additive** scoring system → **log-odds ratio**:

$$S = \sum_i \log \left(\frac{p_{x_i y_i}}{q_{x_i} q_{y_i}} \right) = \sum_i s(x_i, y_i)$$

- $s(a,b)$: log-likelihood ratio of pair (a,b) occurring as an **aligned pair** as opposed to an **unaligned pair** $\rightsquigarrow$ **substitution matrix**.

BLOSUM62 substitution matrix

Ala	4																			
Arg	−1	5																		
Asn	−2	0	6																	
Asp	−2	−2	1	6																
Cys	0	−3	−3	−3	9															
Gln	−1	1	0	0	−3	5														
Glu	−1	0	0	2	−4	2	5													
Gly	0	−2	0	−1	−3	−2	−2	6												
His	−2	0	1	−1	−3	0	0	−2	8											
Ile	−1	−3	−3	−3	−1	−3	−3	−4	−3	4										
Leu	−1	−2	−3	−4	−1	−2	−3	−4	−3	2	4									
Lys	−1	2	0	−1	−3	1	1	−2	−1	−3	−2	5								
Met	−1	−1	−2	−3	−1	0	−2	−3	−2	1	2	−1	5							
Phe	−2	−3	−3	−3	−2	−3	−3	−3	−1	0	0	−3	0	6						
Pro	−1	−2	−2	−1	−3	−1	−1	−2	−2	−3	−3	−1	−2	−4	7					
Ser	1	−1	1	0	−1	0	0	0	−1	−2	−2	0	−1	−2	−1	4				
Thr	0	−1	0	−1	−1	−1	−1	−2	−2	−1	−1	−1	−1	−2	−1	1	5			
Trp	−3	−3	−4	−4	−2	−2	−3	−2	−2	−3	−2	−3	−1	1	−4	−3	−2	11		
Tyr	−2	−2	−2	−3	−2	−1	−2	−3	2	−1	−1	−2	−1	3	−3	−2	−2	2	7	
Val	0	−3	−3	−3	−1	−2	−2	−3	−3	3	1	−2	1	−1	−2	−2	0	−3	−1	4
	Ala	**Arg**	**Asn**	**Asp**	**Cys**	**Gln**	**Glu**	**Gly**	**His**	**Ile**	**Leu**	**Lys**	**Met**	**Phe**	**Pro**	**Ser**	**Thr**	**Trp**	**Tyr**	**Val**

Wikipedia

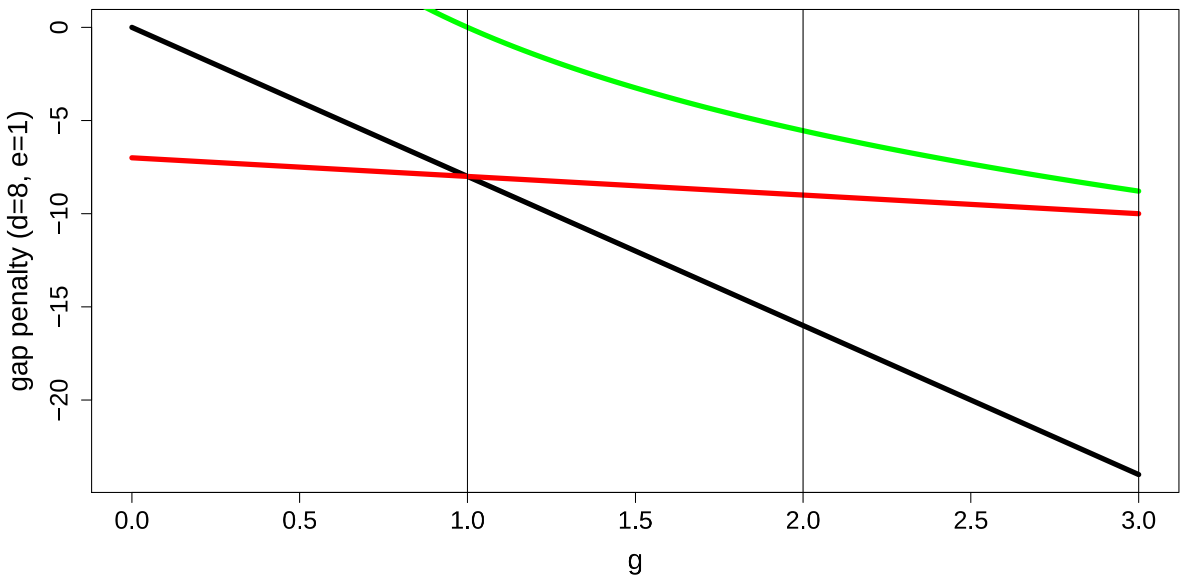
Gap penalties

Gap penalty types for a gap of length g:

- **Linear**: $\gamma(g) = -gd$, with d being the **gap weight**.

- **Affine**: $\gamma(g) = -d - (g-1)e$,
  gap-open penalty d, **gap-extension** penalty e. Usually $e < d$.

- **Convex**: e.g. $\gamma(g) = -d\log(g)$. Each additional space contributes less than the previous space.



Global Alignment: Needleman-Wunsch algorithm

The Global Alignment problem:

INPUT: two sequences $x = x_1 \ldots x_n$ and $y = y_1 \ldots y_m$.

TASK: Find optimal alignment for linear gap penalties $\gamma(g) = -gd$.

Let $F(i,j)$ be the optimal alignment score of the **prefix sequences** $x_{1\ldots i}$ and $y_{1\ldots j}$. A zero index $i = 0$ or $j = 0$ refers to an **empty sequence.** $F(i,j)$ has following properties:

Base conditions:

$$F(i,0) = \sum_{k=1}^{i} -d = -id$$

$$F(0,j) = \sum_{k=1}^{j} -d = -jd, \quad F(0,0) = 0.$$

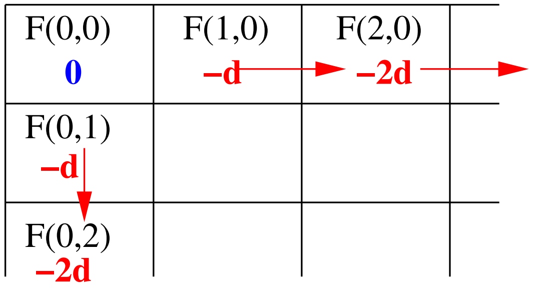
Recurrence relation: for $1 \leq i \leq n,\ 1 \leq j \leq m :$

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

Tabular Computation of Optimal Alignment

Starting from $F(0,0) = 0$, fill the whole matrix $(F)_{ij}$:

for $i = 0$ or $j = 0$, calculate new value from left-hand (upper) value.



F(0,0) 0

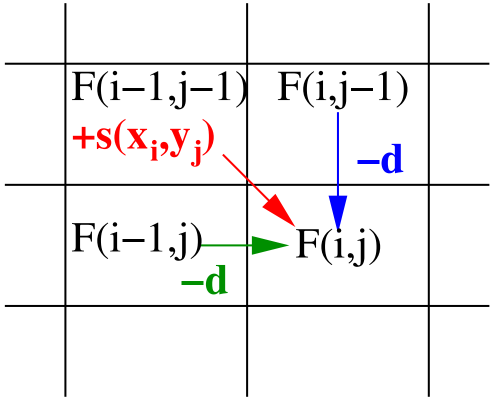
F(1,0) −d

F(2,0) −2d

F(0,1) −d

F(0,2) −2d

for $i, j \geq 1$, calculate the bottom right-hand corner of each square of 4 cells from one of the 3 other cells:



F(i−1,j−1) $+s(x_i,y_j)$

F(i,j−1) −d

F(i−1,j) −d

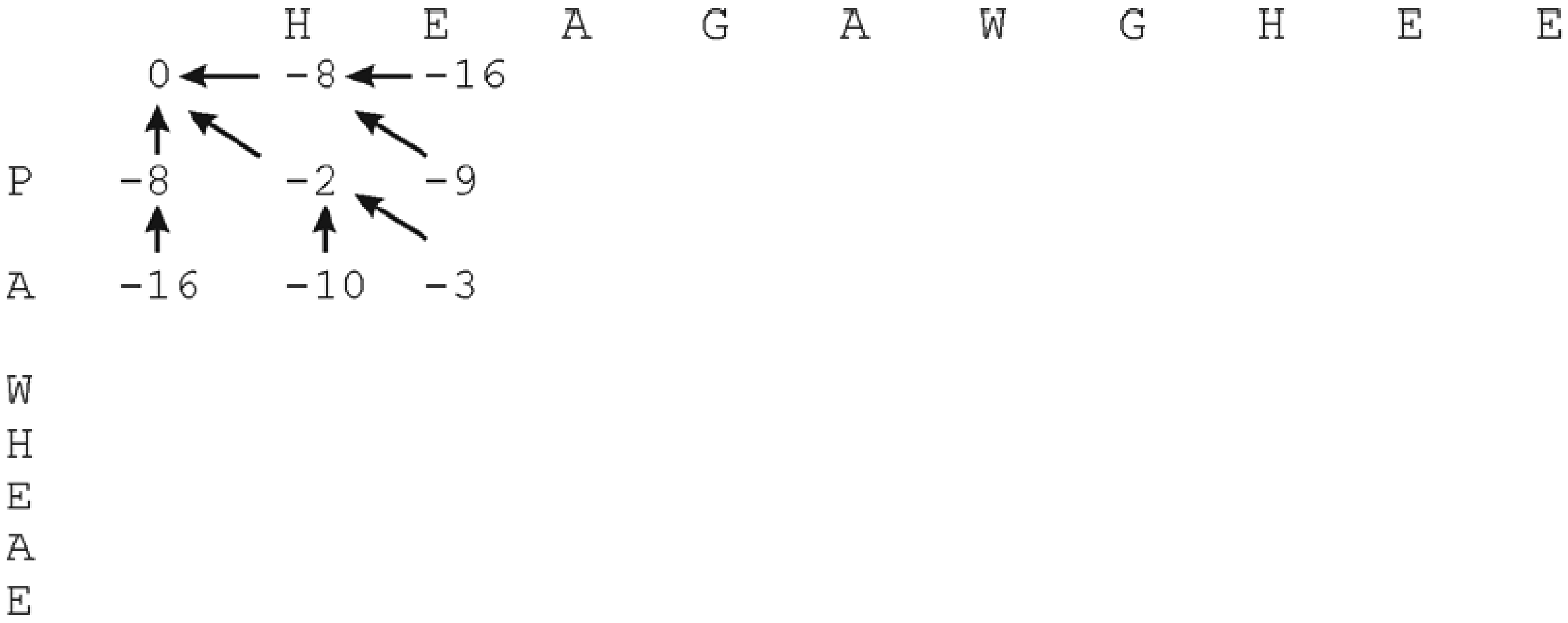
F(i,j)

keep a pointer in each cell back to the cell from which it was derived

$\Rightarrow$ **traceback pointer**.

Global Alignment: Example

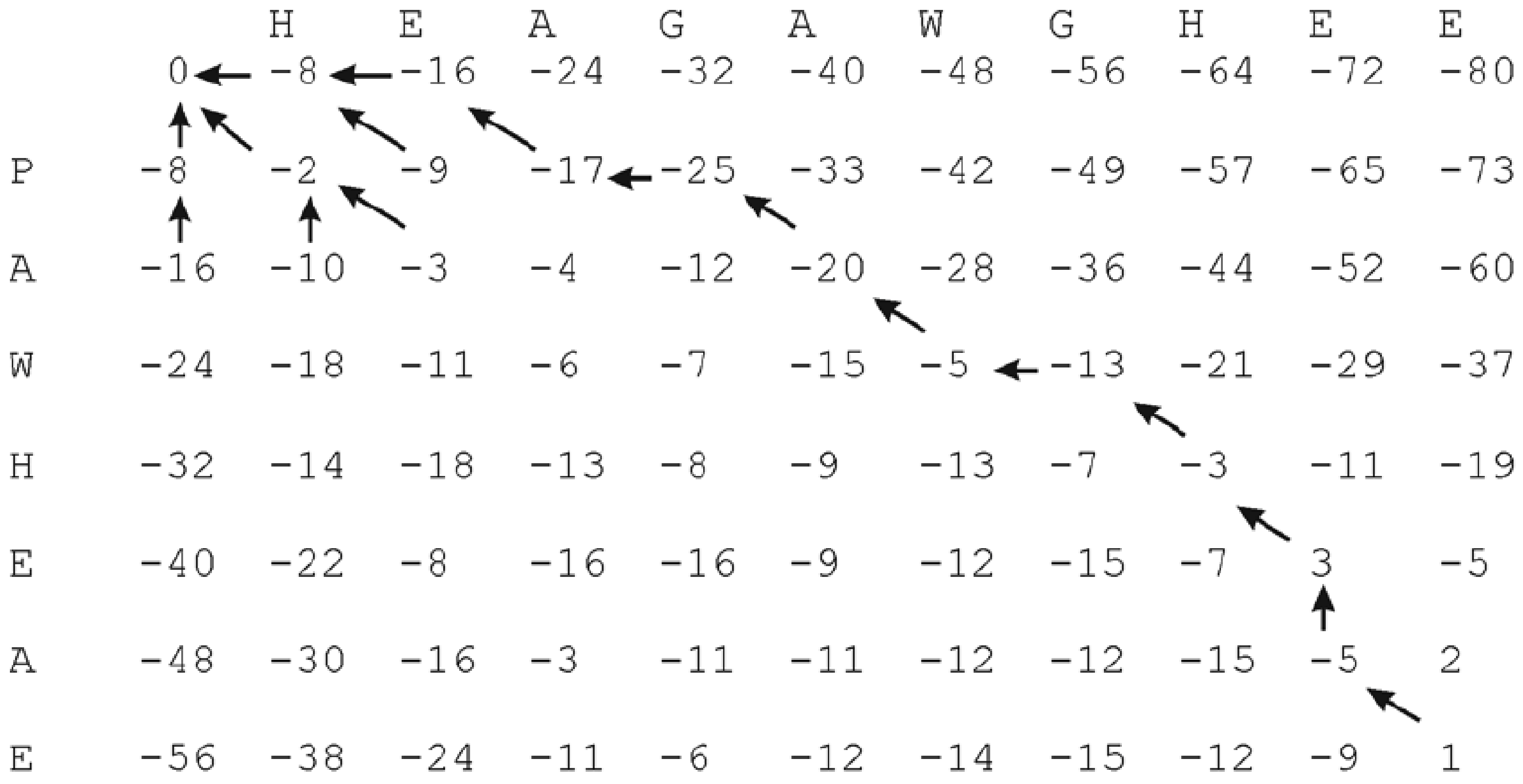
$x =$ HEAGAWGHEE, $y =$ PAWHEAE. Linear gap costs $d = 8$.
Scoring matrix: BLOSUM50



		H	E	A	G	A	W	G	H	E	E
	0	-8	-16								
P	-8	-2	-9								
A	-16	-10	-3								
W											
H											
E											
A											
E											

Durbin et al., Cambridge University Press

Example: traceback procedure



		H	E	A	G	A	W	G	H	E	E
	0	-8	-16	-24	-32	-40	-48	-56	-64	-72	-80
P	-8	-2	-9	-17	-25	-33	-42	-49	-57	-65	-73
A	-16	-10	-3	-4	-12	-20	-28	-36	-44	-52	-60
W	-24	-18	-11	-6	-7	-15	-5	-13	-21	-29	-37
H	-32	-14	-18	-13	-8	-9	-13	-7	-3	-11	-19
E	-40	-22	-8	-16	-16	-9	-12	-15	-7	3	-5
A	-48	-30	-16	-3	-11	-11	-12	-12	-15	-5	2
E	-56	-38	-24	-11	-6	-12	-14	-15	-12	-9	1

```
H  E  A  G  A  W  G  H  E  -  E
-  -  P  -  A  W  -  H  E  A  E
```

Add pair of symbols: $\nwarrow$: (x_i, y_j), $\uparrow$: $(-, y_j)$, $\leftarrow$: $(x_i, -)$

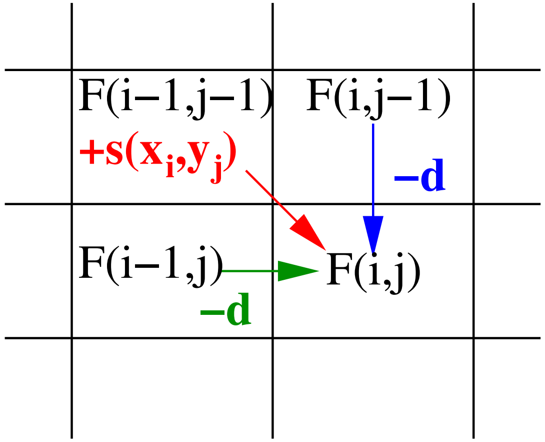
Adapted from Durbin et al., Cambridge University Press

Time and Space Complexity

Theorem. *The time complexity of the Needleman-Wunsch algorithm is $O(nm)$. Space complexity is $O(m)$, if only $F(x,y)$ is required, and $O(nm)$ for the reconstruction of the alignment.*

Proof:

Time: when computing $F(i,j)$, only cells $(i-1,j-1)$, $(i,j-1)$, $(i-1,j)$ are examined $\rightsquigarrow$ constant time. There are $(n+1)(m+1)$ cells $\rightsquigarrow$ $O(nm)$ **time complexity.**



Space : row-wise computation of the matrix: for computing row k, only row $k-1$ must be stored $\rightsquigarrow$ $O(m)$ **space.**

Reconstructing the alignment: all traceback pointers must be stored $\rightsquigarrow$ $O(nm)$ **space complexity.**

Local Alignments

The Local Alignment problem:
INPUT: two sequences $x = x_1, \ldots, x_n$ and $y = y_1, \ldots, y_m$.
TASK: find subsequences a of x and b of y,
whose similarity (=optimal global alignment score) is maximal
(over all such pairs of subsequences).
Assume linear gap penalties $\gamma(g) = -gd$.

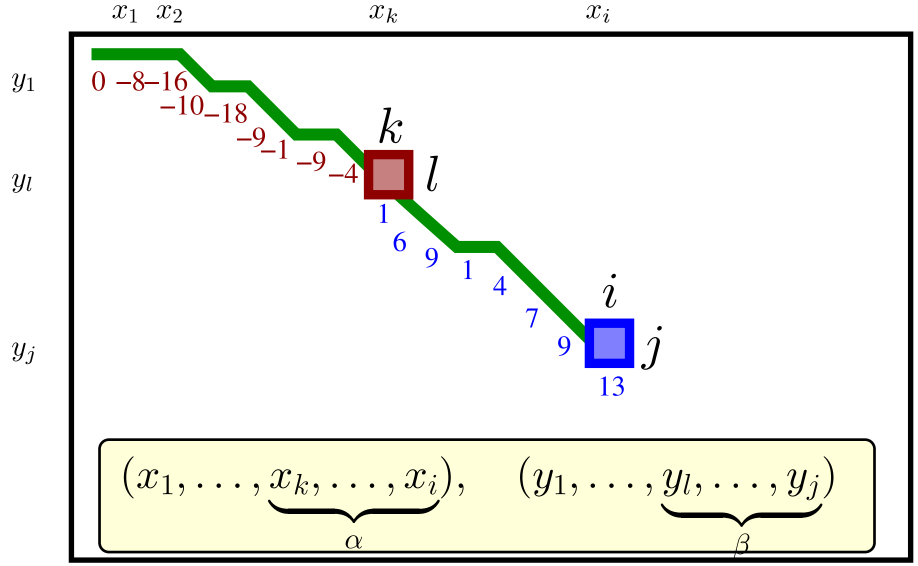
Subsequence = **contiguous** segment of a sequence.

Consider first a simpler problem by **fixing the endpoint** of the subsequences at index pair (i, j):
Local suffix alignment problem: given x, y, i, j, find suffixes α of $x_{1,\ldots,i}$ and β of $y = y_{1,\ldots,j}$ such that their global alignment score is maximal.

$$(x_1, \ldots, \underbrace{x_k, \ldots, x_i}_{\alpha}), \quad (y_1, \ldots, \underbrace{y_l, \ldots, y_j}_{\beta})$$

Local suffix alignments

Consider global alignment path to cell (i,j). Where to start?
Intuition: Indices (k,l) found by following the path back to $(0,0)$, but stopping at the first negative value.



$$(x_1,\ldots,\underbrace{x_k,\ldots,x_i}_{\alpha}), \quad (y_1,\ldots,\underbrace{y_l,\ldots,y_j}_{\beta})$$

Remark: If we consider all solutions (i.e. for all (i,j) pairs), we look at all possible subsequences (no restrictions on α,β)

Maximal solution of local suffix alignment over all pairs (i,j)
= solution of local alignment problem.

Smith-Waterman Algorithm

$F(i, j)$: optimal local suffix alignment for indices i, j.

Global alignment with one **modification:**
Prefixes whose scores are ≤ 0 are **discarded**
$\rightsquigarrow$ alignment can **start anywhere**.

Recurrence relation:

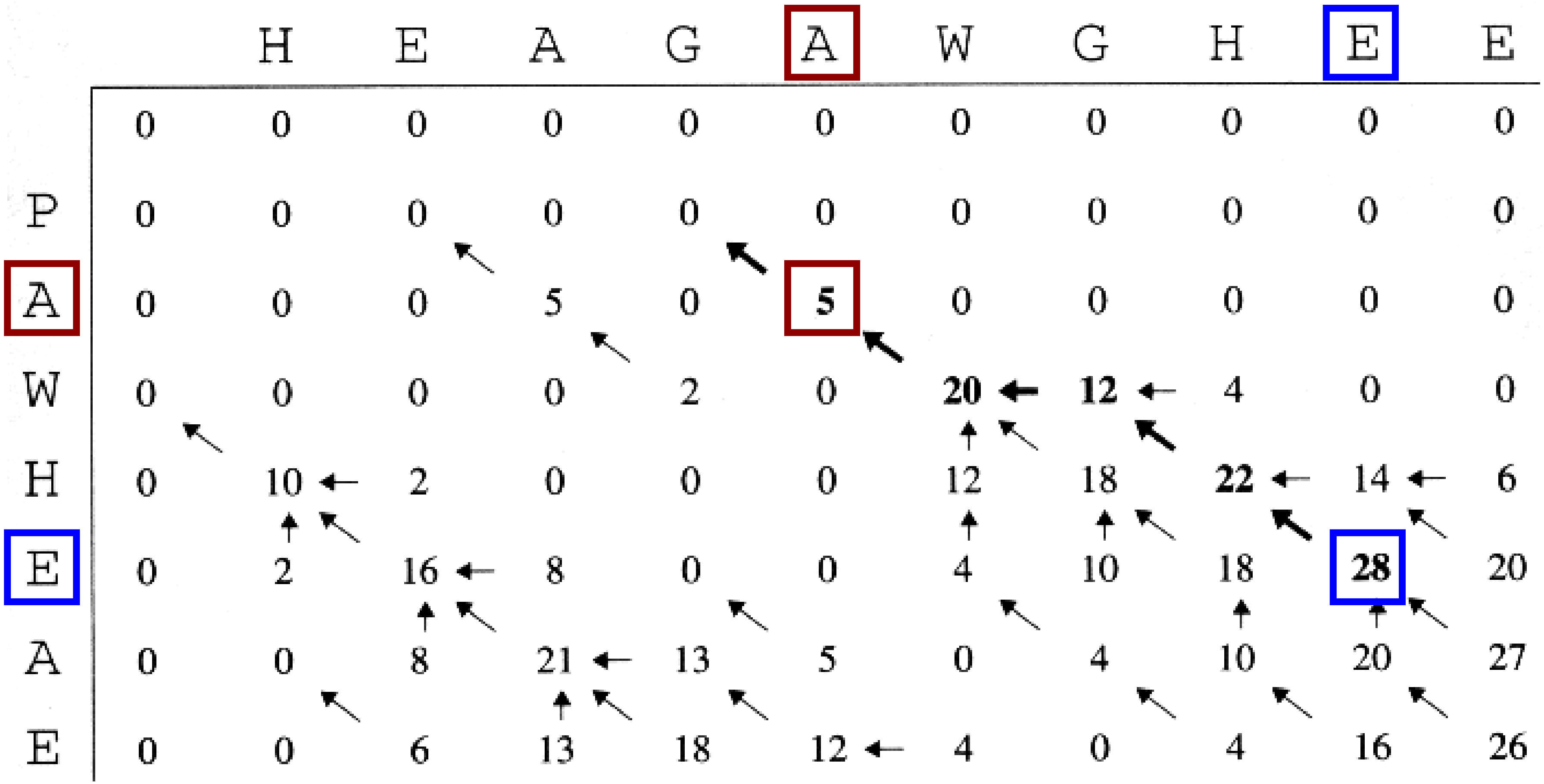
$$F(i, j) = \max \begin{cases} 0 \\ F(i-1, j-1) + s(x_i, y_j) \\ F(i-1, j) - d \\ F(i, j-1) - d \end{cases}$$

Finally, find indices i^* and j^* **after which the similarity only decreases.**
Stop the alignment there.

$$F(i^*, j^*) = \max_{i,j} F(i, j)$$

Traceback...

...starts at highest value until a cell with 0 is reached.



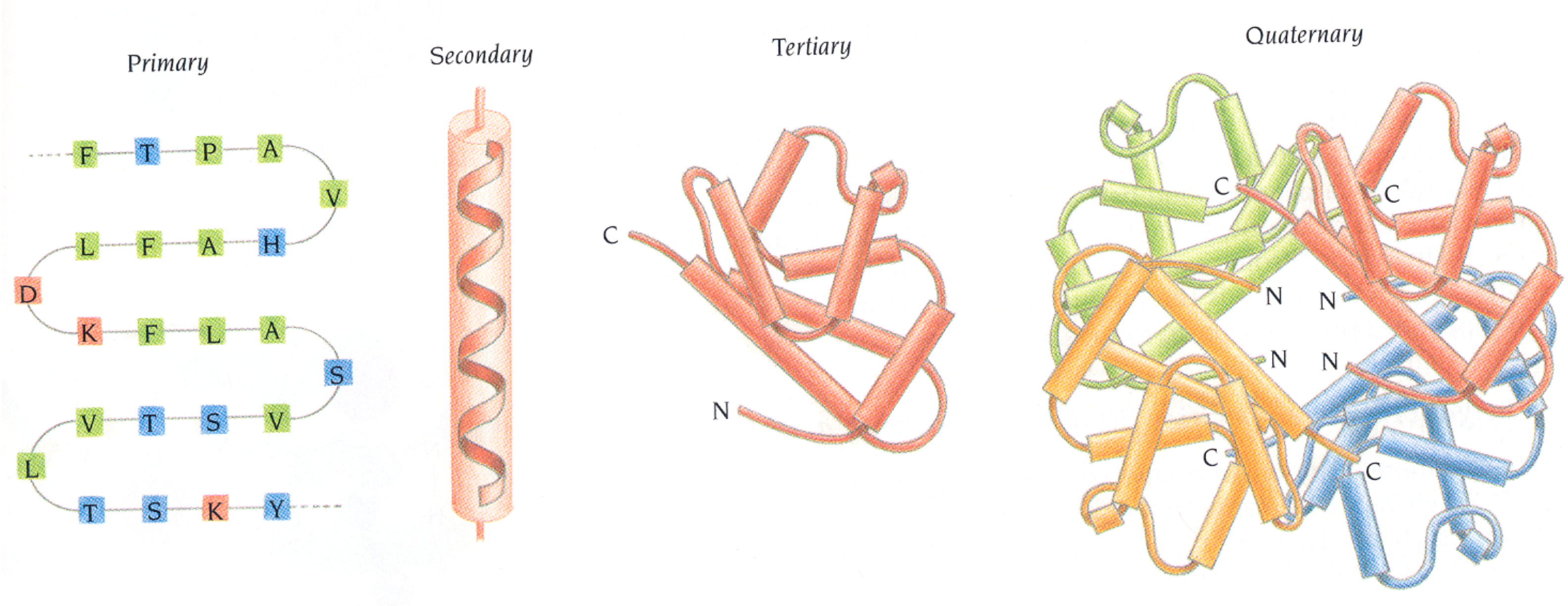
		H	E	A	G	A	W	G	H	E	E
	0	0	0	0	0	0	0	0	0	0	0
P	0	0	0	0	0	0	0	0	0	0	0
A	0	0	0	5	0	**5**	0	0	0	0	0
W	0	0	0	0	2	0	**20** ←	**12** ←	4	0	0
H	0	10 ←	2	0	0	0	12	18	**22** ←	14 ←	6
E	0	2	16 ←	8	0	0	4	10	18	**28**	20
A	0	0	8	21 ←	13	5	0	4	10	20	27
E	0	0	6	13	18	12 ←	4	0	4	16	26

AWGHE
AW-HE

Adapted from Durbin et al., Cambridge University Press. https://doi.org/10.1017/CBO9780511790492.004

Local vs. Global Alignment: Biological Considerations

- Many proteins have **multiple domains**, or modules.
- Some domains are present (with high similarity) in many other proteins
- **Local** alignment can detect similar regions in otherwise dissimilar proteins.



Durbin et al., Cambridge University Press. https://doi.org/10.1017/CBO9780511790492.004