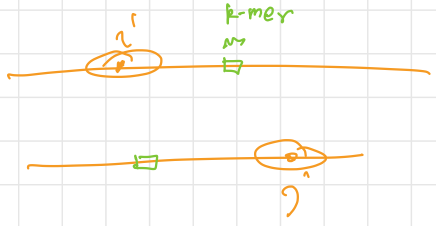
Review:

① Filtration
② sensitivity v.s. speed
③ spaced seed better.

4^{-k}

$\underbrace{11\cdots1}_{k-1}0\ \underbrace{11\cdots1}_{k-1}0\cdots$



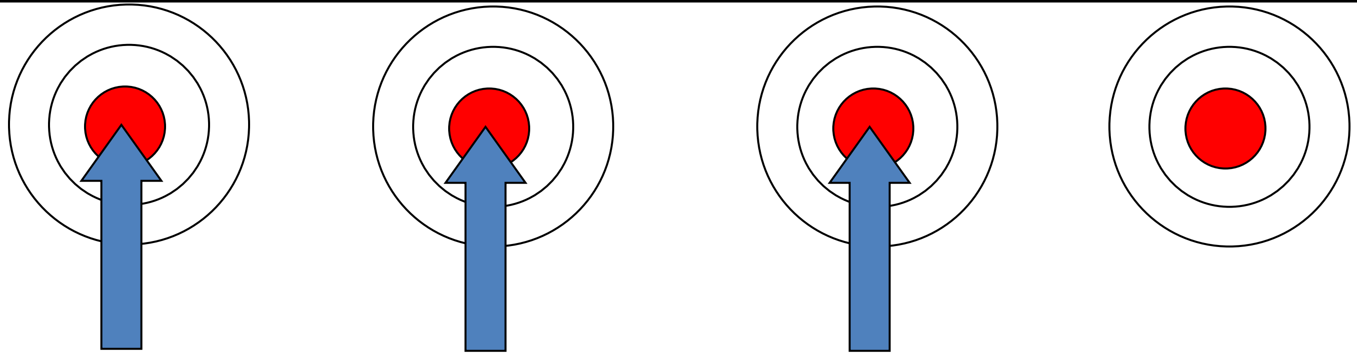
$\underline{11*1}$ spaced seed.

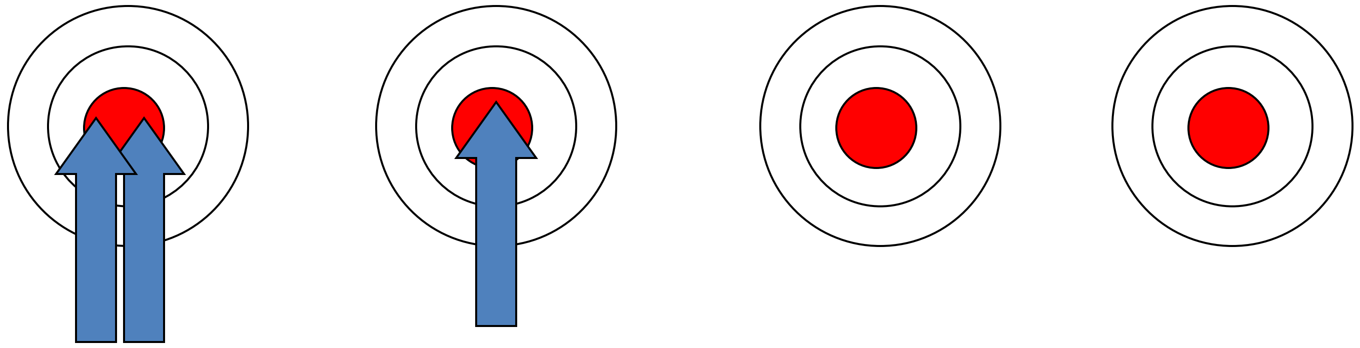
$Pr(\text{hit at } i) = p^w$
$Pr(\text{hit at } j) = p^w$

$\underline{HSP}$

$prob(\text{match}) = p$
$prob(\text{mismatch}) = 1-p.$

Expected number of hits in an HSP.





PH's seed does not overlap much

- PH's seed do not overlap heavily when shifts:

```
111*1**1*1**11*111
 111*1**1*1**11*111   6
  111*1**1*1**11*111   6
   111*1**1*1**11*111   6
    111*1**1*1**11*111   7
     111*1**1*1**11*111   7
      111*1**1*1**11*111   7
             ......
```

- The hits at different positions are independent.
- The probability of having the second hit is $\underline{3^*p^6 + \ldots}$
  - compare to BLAST's seed $\underline{p + p^2 + p^3 + p^4} + \ldots$

Lossless Filtration

- When seeds are short enough and HSP similarity is high enough, lossless filtration is also possible.
- For example, seed 111 can guarantee to match when a sufficiently long HSP has similarity 66.7%.
- Proof: To fail being hit by 111, the HSP must have a mismatch in every 3 adjacent positions.
- On the other hand, 110110110…, which has 66.6% similarity, will fail the seed 111.

Lossless Filtration

$1^a 0^b$ $\quad$ $1^a 0^b$

- Now consider spaced seed 11*1.
- Claim: For any $\epsilon > 0$, seed 11*1 will hit every sufficiently long region with similarity $0.6 + \epsilon$.

proof: $\quad 1^a 0^b \quad$ then $a \leq 3$

$1110\,1^a 0^b$

$110\,1^a 0^b$

$a = 3 \Rightarrow b \geq 2 \qquad$ identity $\leq \frac{3}{5} \qquad \frac{a}{a+b}$

$a = 2 \Rightarrow b \geq 2 \qquad \leq \frac{2}{4}$

$a = 1 \Rightarrow b \geq 1 \qquad \leq \frac{1}{2}$

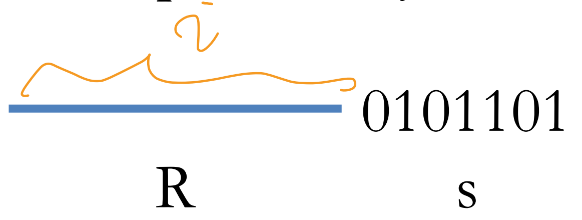
Proof

- Suppose there is a sufficiently long region not hit by 11*1.
- We can break the region into blocks of 1^a0^b. Besides the last block that can have at most three 1s, each of the other blocks is one of the following three cases:
  - 10^b for b>=1
  - 110^b for b>=2
  - 1110^b for b>=2
- In each block, similarity <= 0.6.
- So the long region's similarity is $< 0.6 + \epsilon$.

Compute a Seed's Sensitivity

- R: A probabilistic distribution of HSP, Pr(R[i]=1) = p;
- We want Pr(length-n R is hit by a seed x). |x|=k
- s: A length-k 0-1 string.
- Rs: The concatenation of R and s.
- Let $D[i, s]$ be the probability Rs is hit by x for a length-i R.

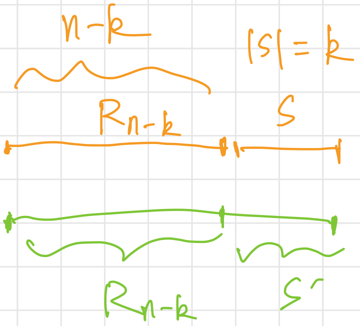
P(length-n R is hit by x).



D[i, s] = Pr(x hits Rs).

- By total probability law, answer is $\sum_s (p(s) \cdot D[n-k, s])$. Note the summation is over all length k binary string s, and $p(s) = p^{\#1\ in\ s}(1-p)^{\#0\ in\ s}$

$D[n-k, s]$



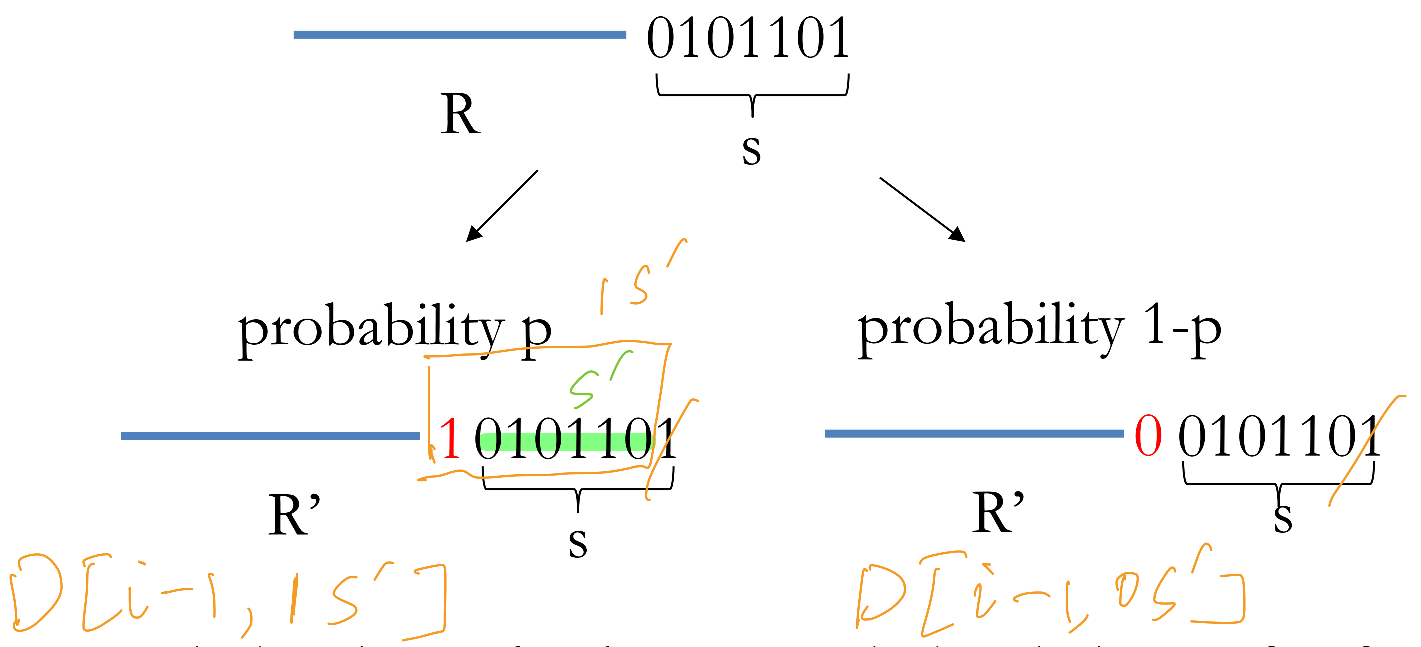
$s' \in \{0,1\}^k$

$$P(R_n \text{ is hit}) = \sum_{s' \in \{0,1\}^k} P(s') \cdot P(R_{n-k}s' \text{ is hit})$$

$$= \sum_{s' \in \{0,1\}^k} p^{\#1 \text{ in } s'} (1-p)^{\#0 \text{ in } s'} \cdot D[n-k, s']$$

Dynamic Programming

- Case I: s is hit by x. Then $D[i, s] = 1$.
- Case II: s is not hit by x:

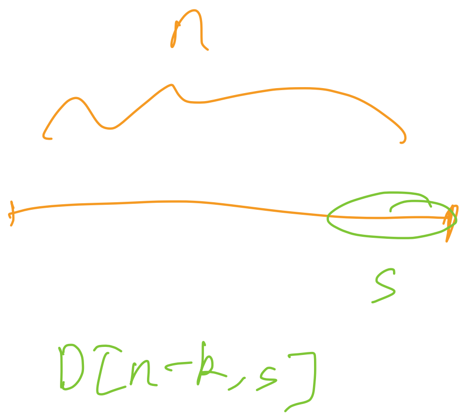


R' is the length-(i-1) distribution. s' is the length-(k-1) prefix of s.

$$D[i, s] = p \cdot D[i-1, 1s'] + (1-p) \cdot D[i-1, 0s']$$

Dynamic Programming

```
• Initialize D[0,s]
• For i from 1 to n
•     for every binary string s   with length k
•         if s is hit by x
•             $D[i,s] = 1$
•         else
•             $D[i,s] = p \cdot D[i-1,1s'] + (1-p) \cdot D[i-1,0s']$
• Return $\sum_s p(s) \cdot D[n-k,s]$
```



Here $p(s) = p^{\#1\ in\ s}(1-p)^{\#0\ in\ s}$.

Time complexity $O(2^k n)$

More efficient algorithm exists (not lectured here). $O(2^{\#0\ in\ s} n)$.

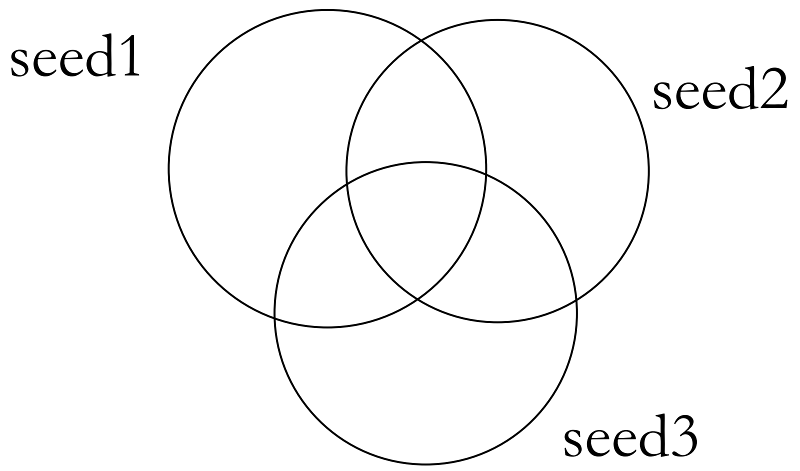
The “algorithm” to select the optimal spaced seed

- Enumerate all spaced seeds with weight 11 and no longer than 18, calculate the sensitivity of each, and output the one with the highest sensitivity.
- This is the ONLY known algorithm that guarantees the finding of optimal seed.
- Many heuristics exist to find suboptimal seeds.

Multiple Seeds – PatternHunter II:

Multiple Spaced Seeds

- Seeds with different shapes can detect different homologies.
  - Some seeds *may* detect more homologies than others. This leads to the use of optimized spaced seed.
  - Can use several seeds simultaneously to hit more homologies
    - Approaching 100% sensitive homology search



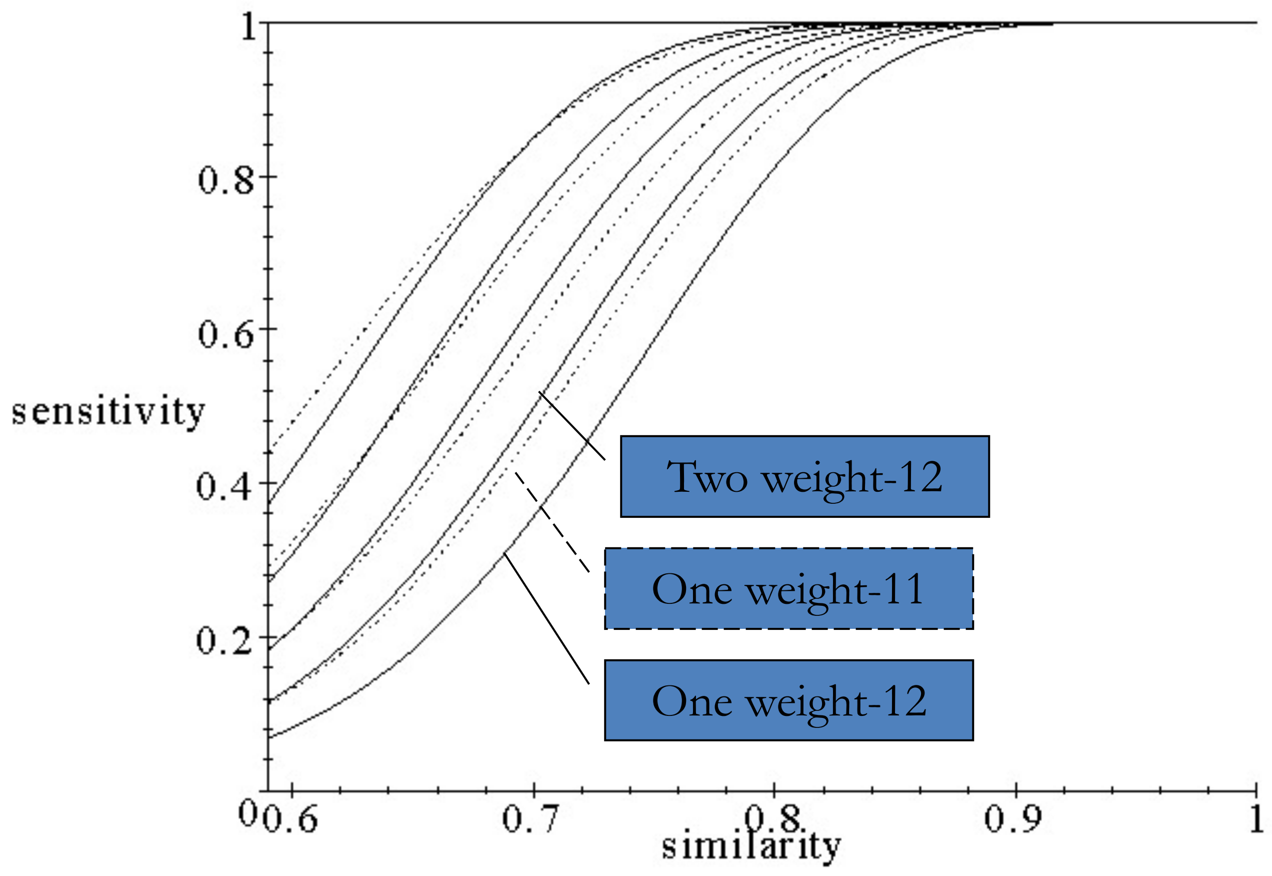
Multiple Seeds Example

(homology identity = 0.7, homology length=64)

```
111*11**1*11*1*111
1111***1***1**11*1*111
11**11*1**1*1***11*111
111*1***1111**1***11*1
```

- To use multiple seeds, one only needs to search multile times with different seeds, and combine results. Of course, you can search with them simultaneously.
- In either case, this slows down approximately k times if k seeds are used.
- Is it worth it? How does it compare with using one shorter seed?

Simulated sensitivity curves:



- Solid curves: Multiple (1, 2, 4, 8, 16) weight-12 spaced seeds.
- Dashed curves: Optimal spaced seeds with weight = 11, 10, 9, 8.

- Typically, "Doubling the seed number" gains better sensitivity than "decreasing the weight by 1".

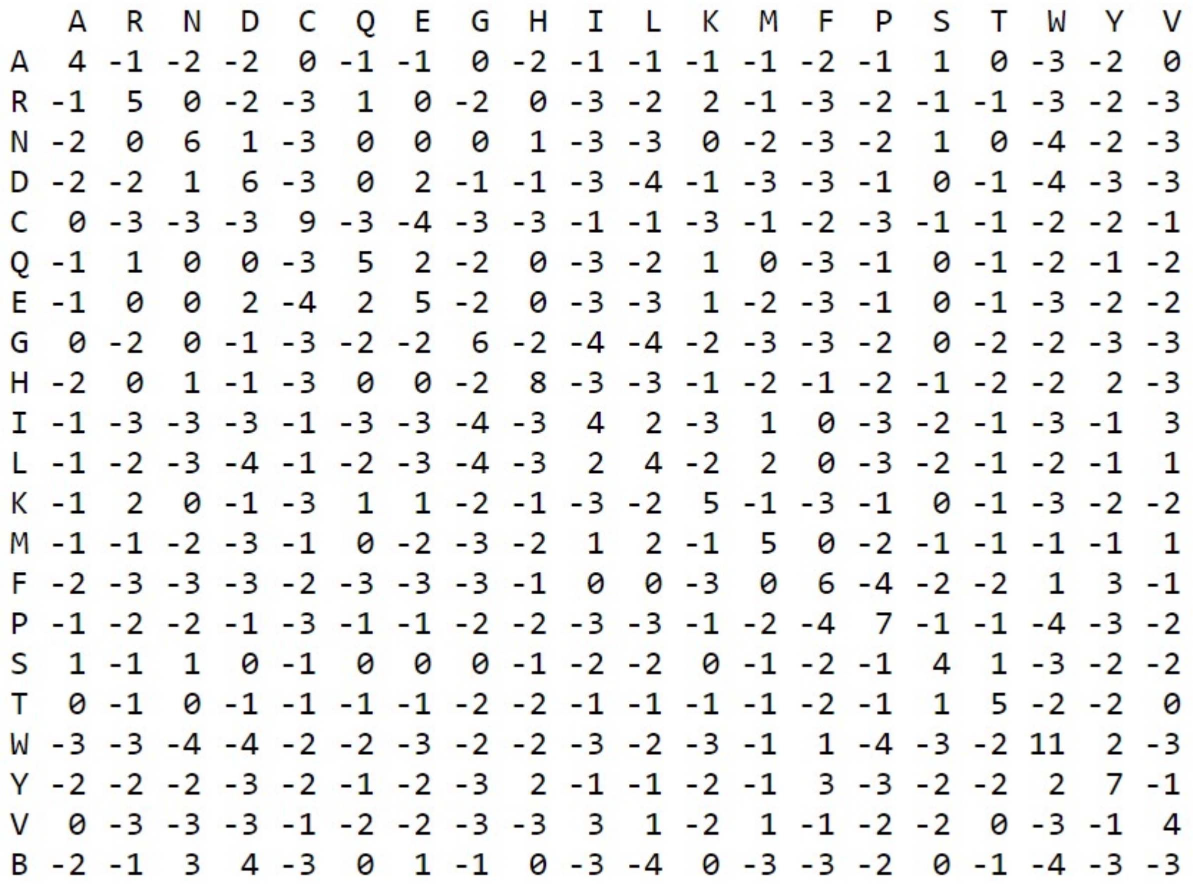
Seeding for Proteins - BLASTP

- With nucleotides, we're requiring *k* positions with exact matches.

- For proteins, that's not really reasonable: some amino acids mutate to another one very often.
- So BLASTP looks for 3- or 4-letter protein sequences that are "very close" to each other, and then builds matches from them.
- Where very close ➔ total BLOSUM score in the short window is at least +13 (or +11 for 3 mer).

Excercise

- For query RRR, threshold 11, what are the other 3-mers that can generate hits?



	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	4	-1	-2	-2	0	-1	-1	0	-2	-1	-1	-1	-1	-2	-1	1	0	-3	-2	0
R	-1	5	0	-2	-3	1	0	-2	0	-3	-2	2	-1	-3	-2	-1	-1	-3	-2	-3
N	-2	0	6	1	-3	0	0	0	1	-3	-3	0	-2	-3	-2	1	0	-4	-2	-3
D	-2	-2	1	6	-3	0	2	-1	-1	-3	-4	-1	-3	-3	-1	0	-1	-4	-3	-3
C	0	-3	-3	-3	9	-3	-4	-3	-3	-1	-1	-3	-1	-2	-3	-1	-1	-2	-2	-1
Q	-1	1	0	0	-3	5	2	-2	0	-3	-2	1	0	-3	-1	0	-1	-2	-1	-2
E	-1	0	0	2	-4	2	5	-2	0	-3	-3	1	-2	-3	-1	0	-1	-3	-2	-2
G	0	-2	0	-1	-3	-2	-2	6	-2	-4	-4	-2	-3	-3	-2	0	-2	-2	-3	-3
H	-2	0	1	-1	-3	0	0	-2	8	-3	-3	-1	-2	-1	-2	-1	-2	-2	2	-3
I	-1	-3	-3	-3	-1	-3	-3	-4	-3	4	2	-3	1	0	-3	-2	-1	-3	-1	3
L	-1	-2	-3	-4	-1	-2	-3	-4	-3	2	4	-2	2	0	-3	-2	-1	-2	-1	1
K	-1	2	0	-1	-3	1	1	-2	-1	-3	-2	5	-1	-3	-1	0	-1	-3	-2	-2
M	-1	-1	-2	-3	-1	0	-2	-3	-2	1	2	-1	5	0	-2	-1	-1	-1	-1	1
F	-2	-3	-3	-3	-2	-3	-3	-3	-1	0	0	-3	0	6	-4	-2	-2	1	3	-1
P	-1	-2	-2	-1	-3	-1	-1	-2	-2	-3	-3	-1	-2	-4	7	-1	-1	-4	-3	-2
S	1	-1	1	0	-1	0	0	0	-1	-2	-2	0	-1	-2	-1	4	1	-3	-2	-2
T	0	-1	0	-1	-1	-1	-1	-2	-2	-1	-1	-1	-1	-2	-1	1	5	-2	-2	0
W	-3	-3	-4	-4	-2	-2	-3	-2	-2	-3	-2	-3	-1	1	-4	-3	-2	11	2	-3
Y	-2	-2	-2	-3	-2	-1	-2	-3	2	-1	-1	-2	-1	3	-3	-2	-2	2	7	-1
V	0	-3	-3	-3	-1	-2	-2	-3	-3	3	1	-2	1	-1	-2	-2	0	-3	-1	4
B	-2	-1	3	4	-3	0	1	-1	0	-3	-4	0	-3	-3	-2	0	-1	-4	-3	-3

How to implement that?

- With BLASTP:
  - Build an automaton that reflects all string close to short strings in T (the short sequence)
  - Scan S (the longer sequence), looking for matches.
- We do not study the classic ways to match multiple patterns efficiently. If interested, you can read at https://en.wikipedia.org/wiki/Aho%E2%80%93Corasick_algorithm

A Simpler Way

- There is another way:

1) For every 3-mer, find all "neighboring" 3-mers that, score at least +11 (or whatever). Build these into a data structure NeighborList.
2) Build a hash table H for S of its 3-mers, just like for the nucleotide case
3) For every 3-mer *x* in T, retrieve all neighbors from NeighborList. For each neighbor, query H to find hits in S.

NeighborList is a small structure: there are only 8000 3-mers.