Opinionated Lessons in Statistics

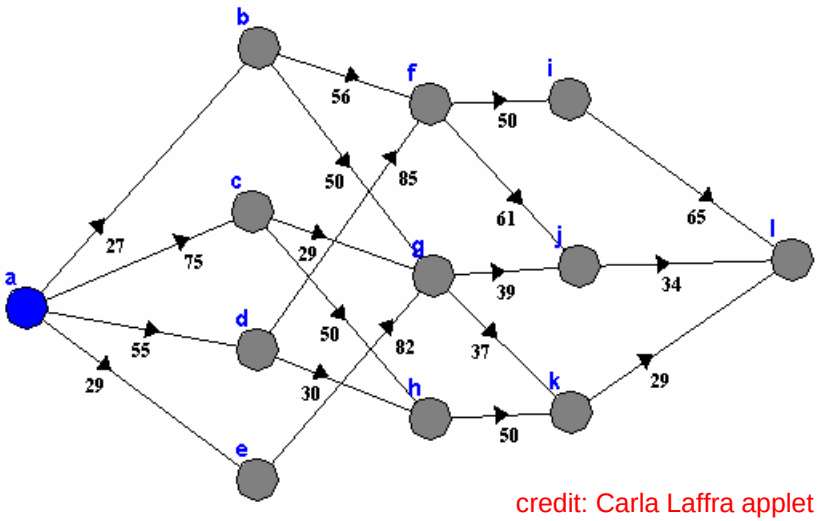
by Bill Press

#52 Dynamic Programming

Dynamic Programming

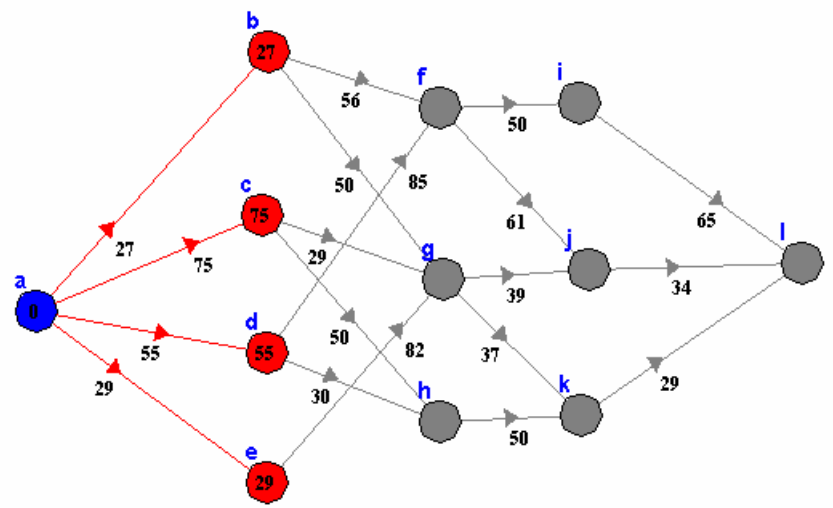
Shortest path algorithm
a.k.a. Bellman or Dijkstra or Viterbi algorithm
(an algorithm too good to be discovered just once!)

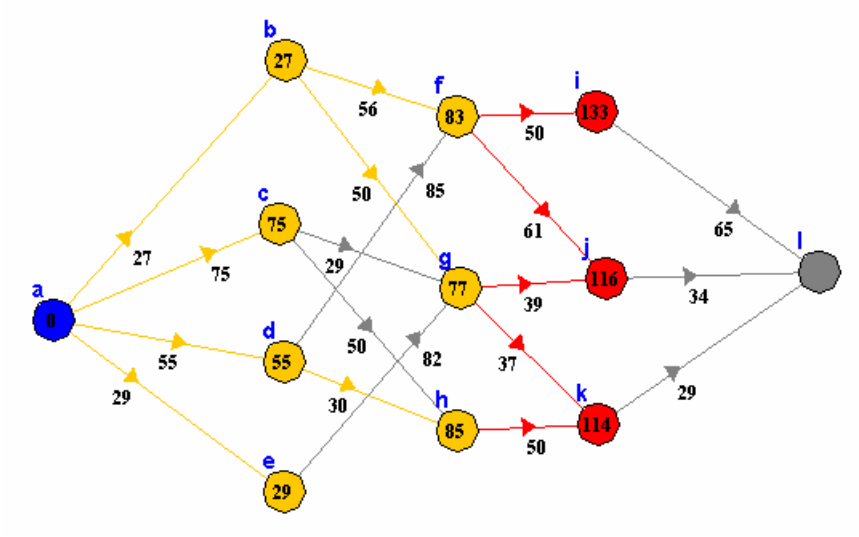
What is the shortest path from a to l, where the edge values signify lengths?

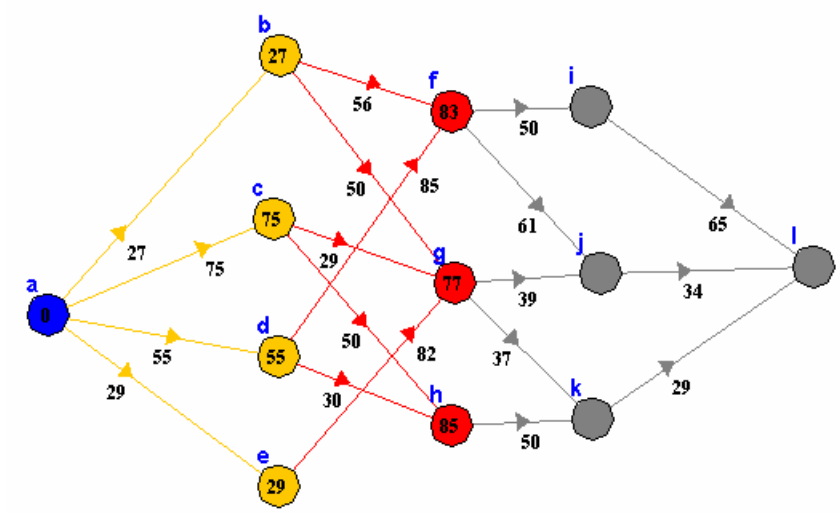


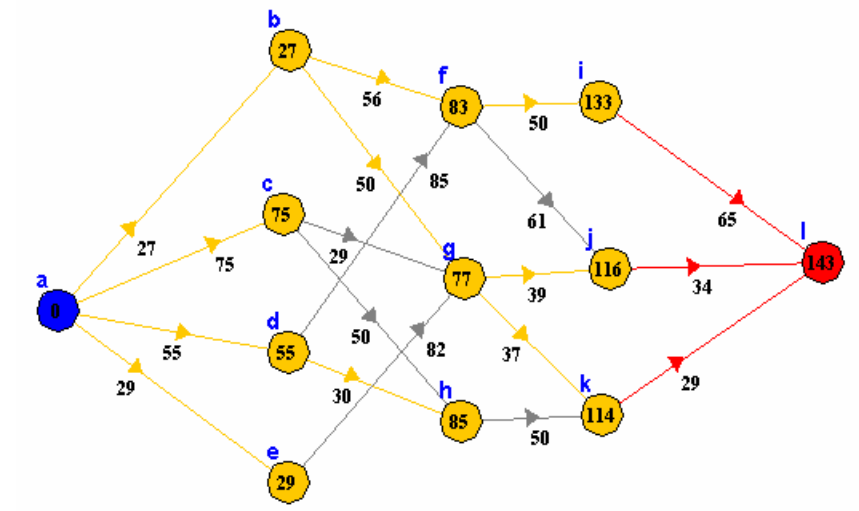
This is general on any DAG (directed acyclic graph = without loops).
Notice that the sums could be products (using logs), and thus
products of probabilities, and thus shortest path = most likely path.

Forward pass: Recursively label nodes by the length of the best path to them

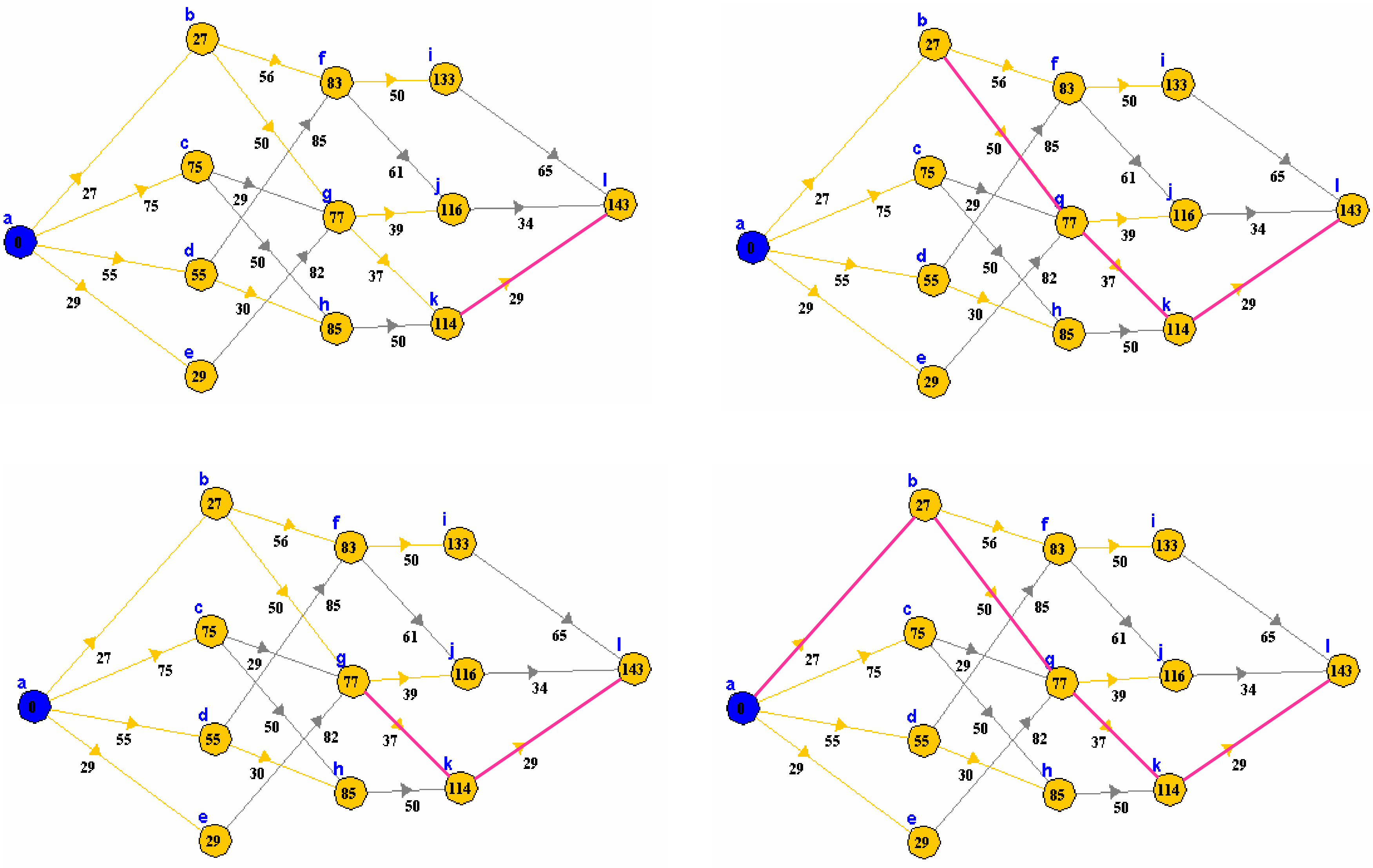








Backward pass: Reconstruct the path that gave those best scores



Maximum likelihood traversal of a graph (trellis) is the basis of the famous Viterbi soft-decision decoding algorithm

Example:

the code:

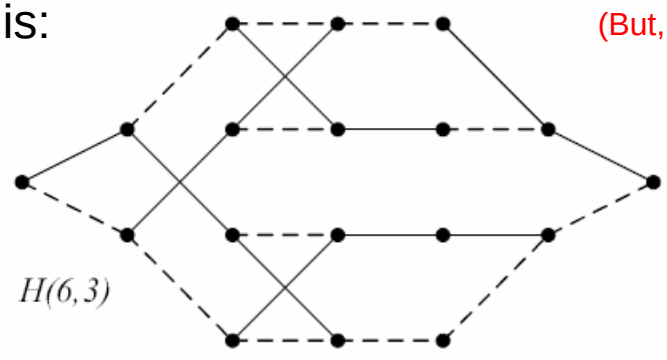
(6,3) Shortened Hamming	
message	codeword
000	000000
001	001110
010	010101
011	011011
100	100011
101	101101
110	110110
111	111000

Hamming distance 3: any pair of codewords differ by at least 3 bits

What would be "hard" instead of "soft" decode? Call bits as zero or one, then look for closest codeword. Will be correct message if 0 or 1 bit is wrong (1 bit error correction).

All codes have trellises. However, it is not always easy to find the "minimal trellis" (smallest maximum expansion). And even the minimal trellis may not be small enough to be practical. Lots of great theory in this! (But, alas, not within the scope of this course.)

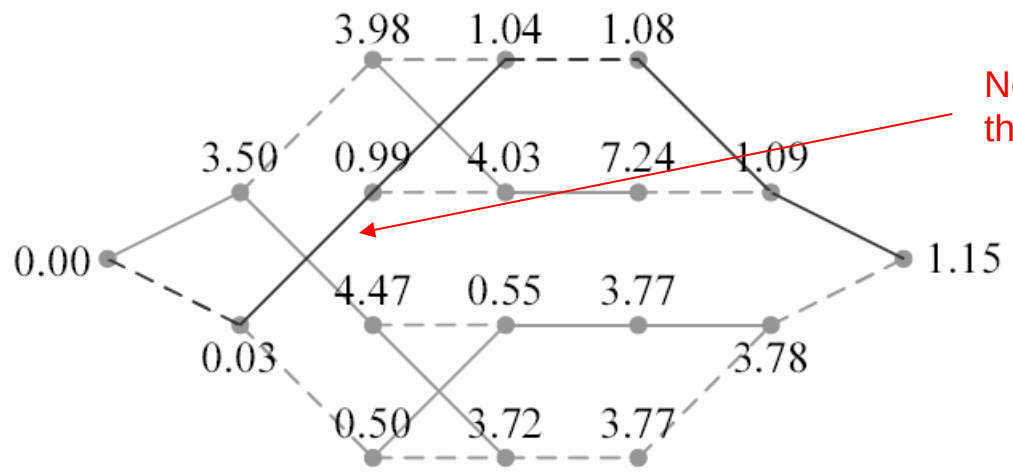
the trellis:



Now suppose that you receive one (noisy) 6-bit codeword, and assign probabilities to what you saw on each bit (hence the word "soft")

bit	1	2	3	4	5	6
Prob(0) =	.97	.62	.04	.96	.01	.06
Prob(1) =	.03	.38	.96	.04	.99	.94
–log[Prob(0)] =	0.03	0.47	3.22	0.04	4.61	2.81
–log[Prob(1)] =	3.50	0.96	0.05	3.21	0.01	0.06

Do the shortest path algorithm



Notice how the shortest path overrides the more likely assignment of the 2nd bit.

Now ready to make a "hard" decision:
Codeword is 011011 ⇒ message is 011

Needleman-Wunsch algorithm cleverly turns string alignment into a problem in dynamic programming

		G	A	A	T	T	C	A	G	T	T	A
	0	0	0	0	0	0	0	0	0	0	0	0
G	0											
G	0											
A	0											
T	0											
C	0											
G	0											
A	0											

credit: Eric Rouchka tutorial

Every path from the upper-left to the lower right by "right/down/diagonal moves" is an alignment, and vice versa.

Right or down moves are gaps.

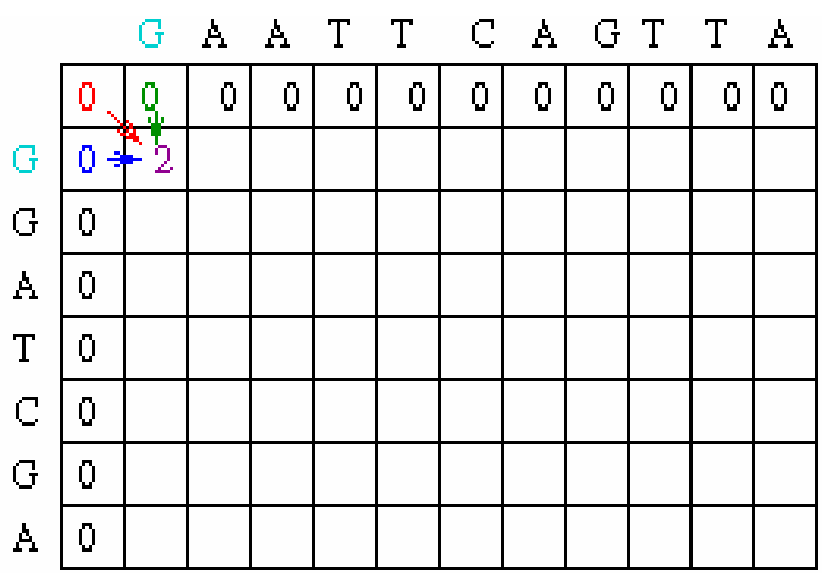
Diagonal moves are matches or mis-matches.

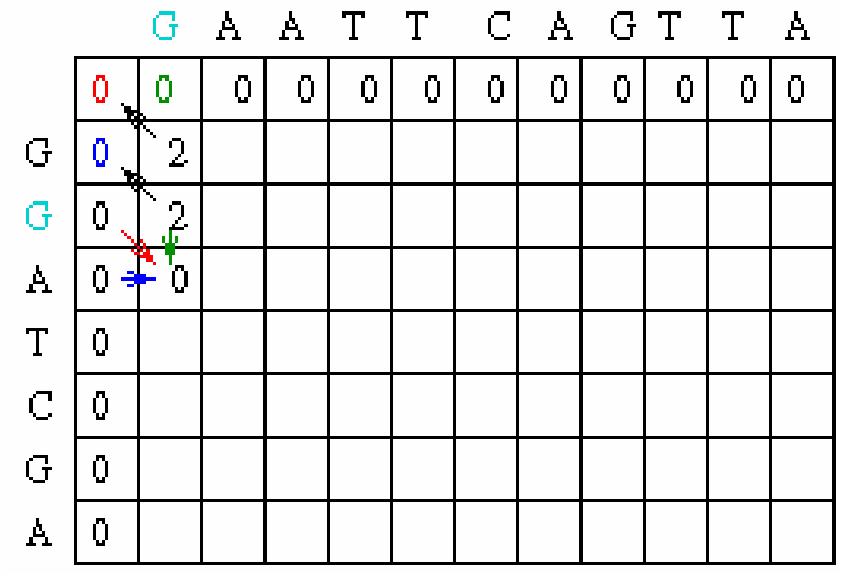
Can assign points to each kind of move.

Example: match= 2, mis-match= –1, gap= –2, initial offset = 0

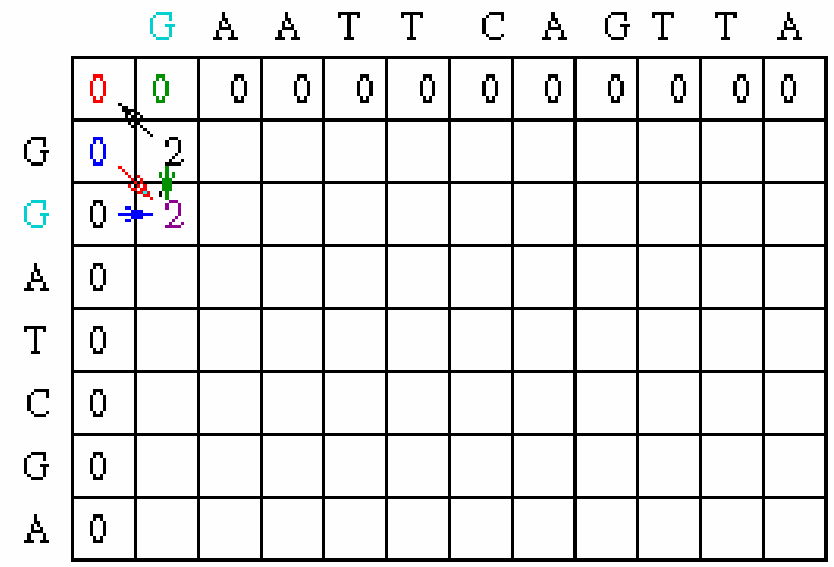
Forward pass: Label each box by the best-path score to it.

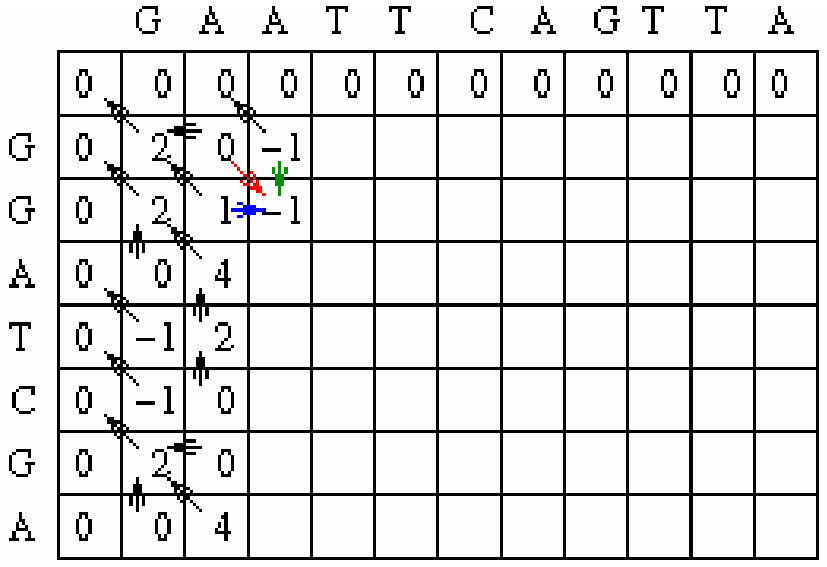
match= 2, mis-match= –1, gap= –2, initial offset = 0



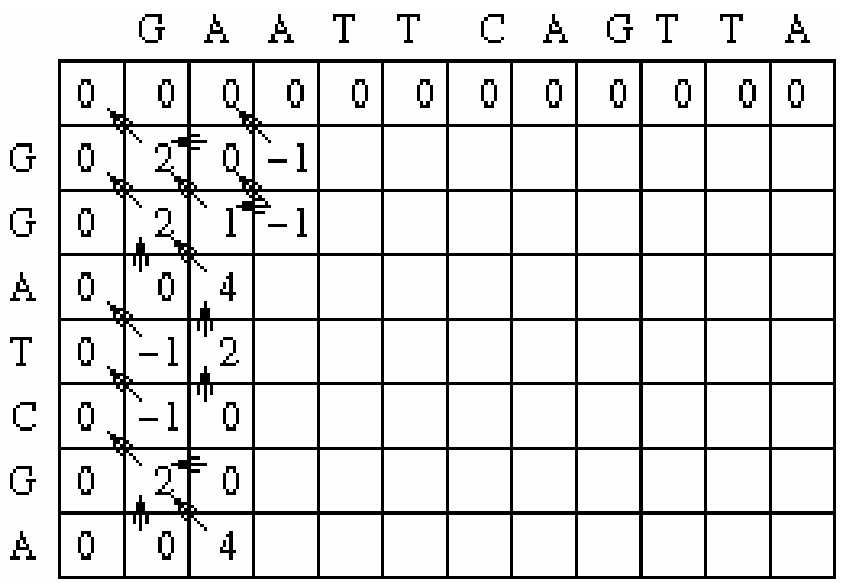


sometimes you can have a tie:



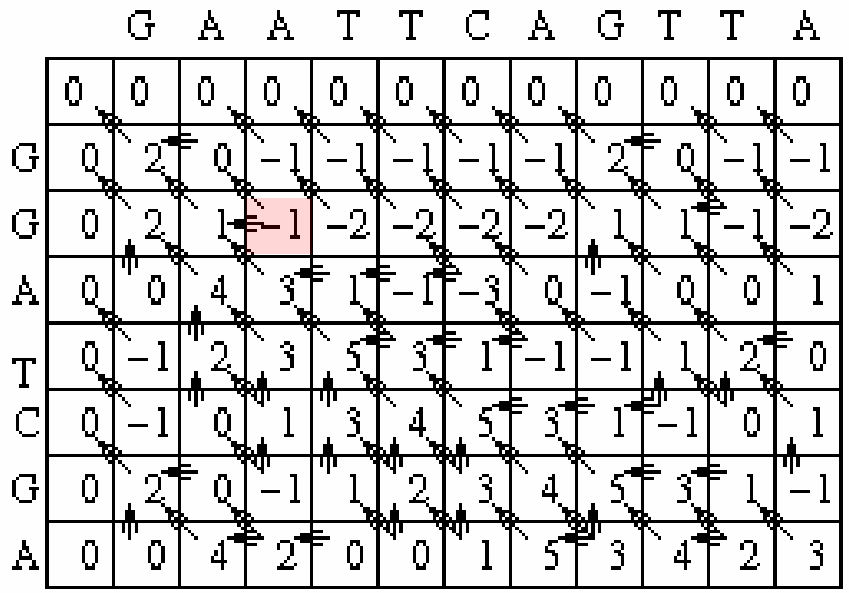


credit: Eric Rouchka tutorial



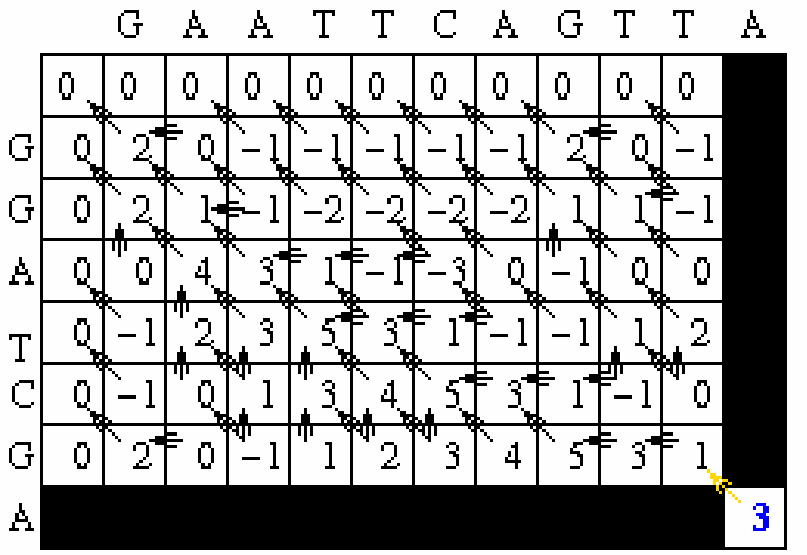
Note that we are keeping track of the back-arrows. Takes a bit of extra storage. It's just as easy to recompute them on the fly along the backward pass, only *O(N).*

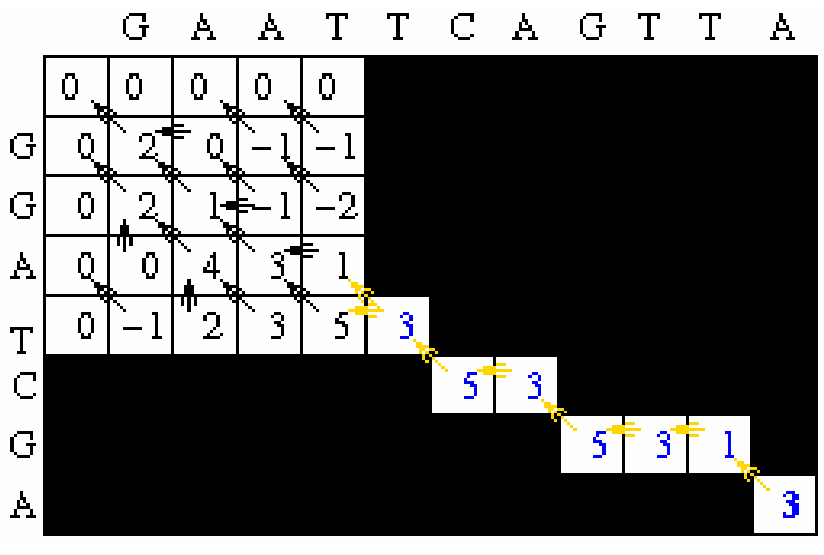
rest of the forward pass:



Note the tie: two back arrows. It's a design decision whether to keep track of more than one. (You don't have to.)

backward pass:

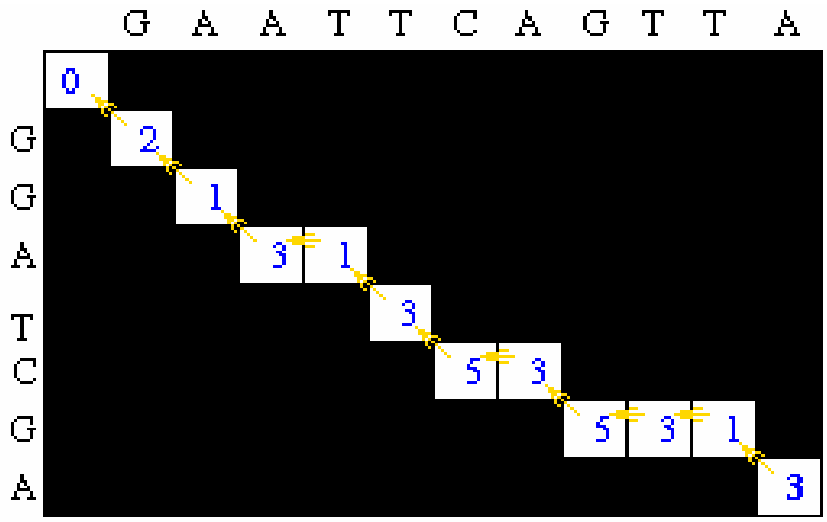





```
A
|
A
```

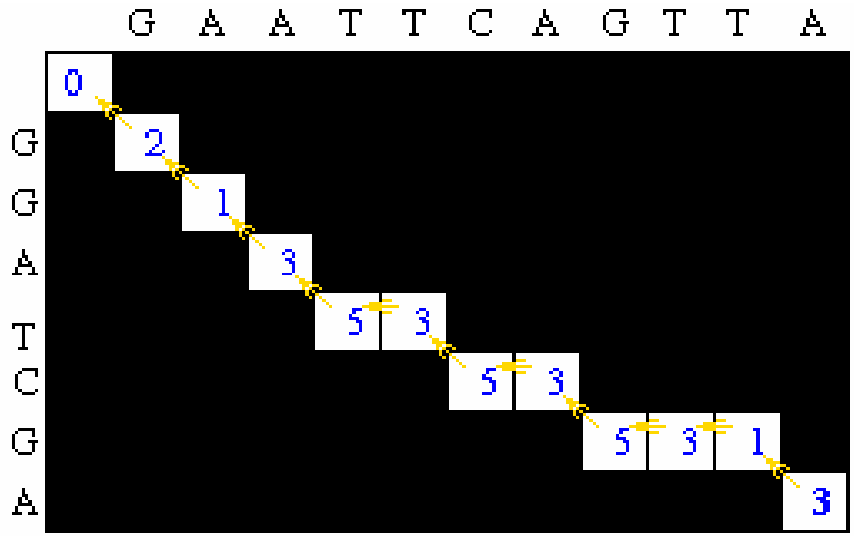
```
T C A G T T A
| |   |     |
T C _ G _ _ A
```

As shown, and just for simplicity, this example forces alignment of the last bases. That is, we allowed slides at the start, but not at the end. To allow them at the end, and an extra empty row and extra empty column column. Then, if you drop off the bottom or right of the matrix above, you'll have to do explicit gaps to get to the new corner. You can either penalize these gaps or not, as you wish.




```
G A A T T C A G T T A
|   |   |   |   |   |
G G A _ T C _ G _ _ A
```

An equally good alternative solution, due to the tie, is:




```
G A A T T C A G T T A
|   |   |   |   |   |
G G A T _ C _ G _ _ A
```

NR3's stringalign function is a simple implementation of Needleman-Wunsch

For real data, there are standard programs with better (but related) algorithms, and which also do multiple alignments (a much harder problem).
ClustalW is most widely used.

(this is "truth" from a fancier multiway alignment program)

```
#include "nr3.h"
#include "stringalign.h"

char a_hg18[]    = "--CCAGGCTGAGCCGC---TGTGGAAGGGGAGAGTGGGTCTTGGTGCTCCATCCCAAACCTGCTGCAGCCGT----GGGTGAGGTCACC-";
char a_rheMac2[] = "GGCCAGGCTGAGCTGC---TGTGGAAG--------GGGTCTTGGTTCTCCATCTGAAACCTGCTGCAGCCGT----GGGTGAGGTCACC-";
char a_equCab1[] = "--------------TC---TGGGTAAG-GTGAAGTGGGCCTTGGCGCCCTACCCAGAATCCTCTGTAACTGTG---GGGTGAGGTCATGG";

void stripalign(char *in, char *out) {
     char *a=in, *b=out;
     while (*a) {
          if (*a != '-') {*b = *a; b++;}
          a++;
     }
     *b = 0; // terminating null
}

void test(char *ans, char *bns) {
     char ain[1024], bin[1024], aout[1024], bout[1024], summary[1024];
     stripalign(ans,ain);
     stripalign(bns,bin);
     printf("%s\n%s\n\n",ain,bin);
     stringalign(ain,bin,1.,3.,0.2,aout,bout,summary);
     printf("%s\n%s\n%s\n\n",aout,bout,summary);
}

int main() {
     test(a_hg18,a_rheMac2);
     test(a_hg18,a_equCab1);
     return 0;
}
```

Input

```
CCAGGCTGAGCCGCTGTGGAAGGGGAGAGTGGGTCTTGGTGCTCCATCCCAAACCTGCTGCAGCCGTGGGTGAGGTCACC
GGCCAGGCTGAGCTGCTGTGGAAGGGGTCTTGGTTCTCCATCTGAAACCTGCTGCAGCCGTGGGTGAGGTCACC
```

Result

```
  CCAGGCTGAGCCGCTGTGGAAGGGGAGAGTGGGTCTTGGTGCTCCATCCCAAACCTGCTGCAGCCGTGGGTGAGGTCACC
GGCCAGGCTGAGCTGCTGTGG      A AG GGGTCTTGGTTCTCCATCTGAAACCTGCTGCAGCCGTGGGTGAGGTCACC
  ===========!=======      = == ==========!=======!!==============================
```

Input

It would be possible within the DP framework, and biologically better, to penalize non-consecutive gaps more than consecutive ones.

```
CCAGGCTGAGCCGCTGTGGAAGGGGAGAGTGGGTCTTGGTGCTCCATCCCAAACCTGCTGCAGCCGTGGGTGAGGTCACC
TCTGGGTAAGGTGAAGTGGGCCTTGGCGCCCTACCCAGAATCCTCTGTAACTGTGGGGTGAGGTCATGG
```

Result

```
CCAGGCTGAGCCGCTGTGGAAGGGGAGAGTGGGTCTTGGTGCTCCATCCCAAACCTGCTGCAGCCGTGG GTGAGGTCACC
            TCTGGGTAAGGTGA AGTGGGCCTTGGCGCCCTACCCAGAATCCTCTGTAACTGTGGGGTGAGGTCATGG
            !===!=!====!== ======!=====!==!=!=!==!!==!=!!===!=!=!==== =========!!
```