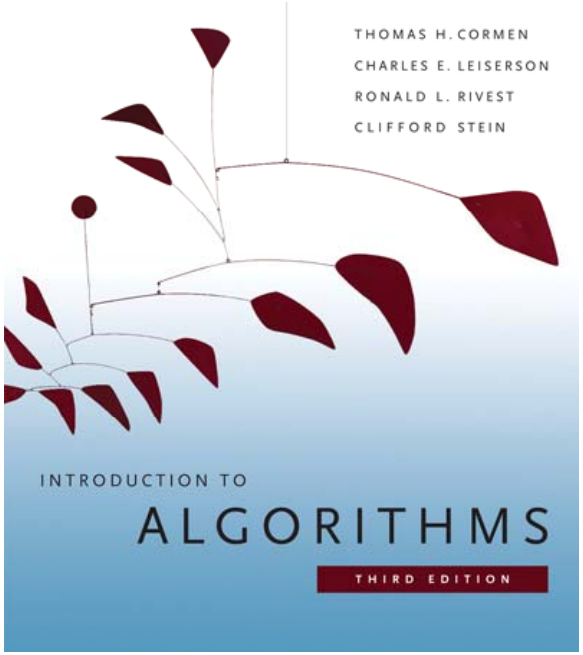
# 6.006- *Introduction to Algorithms*

## *Lecture 18*




## *Dynamic Programming I*

**Prof. Manolis Kellis**

**CLRS 15.3, 15.4**

# Course VI - 6.006 – Module VI – This is it

| Unit | Pset | Week | Date | | Lecture (Tuesdays and Thursdays) | | Recitation (Wed and Fri) |
|---|---|---|---|---|---|---|---|
| Intro | PS1 | 1 | Tue Feb 01 | 1 | Introduction and Document Distance | 1 | Python and Asymptotic Complexity |
| Binary | Out: 2/1 | | Thu Feb 03 | 2 | Peak Finding Problem | 2 | Peak Finding correctness & analysis |
| Search | Due: Mon 2/14 | 2 | Tue Feb 08 | 3 | Scheduling and Binary Search Trees | 3 | Binary Search Tree Operations |
| Trees | HW lab: Sun 2/13 | | Thu Feb 10 | 4 | Balanced Binary Search Trees | 4 | Rotations and AVL tree deletions |
| Hashing | PS2 Out: 2/15 | 3 | Tue Feb 15 | 5 | Hashing I : Chaining, Hash Functions | 5 | Hash recipes, collisions, Python dicts |
| | Due: Mon 2/28 | | Thu Feb 17 | 6 | Hashing II : Table Doubling, Rolling Hash | 6 | Probability review, Pattern matching |
| | HW lab:Sun 2/27 | 4 | Tue Feb 22 | - | President's Day - Monday Schedule - No Class | - | No recitation |
| | | | Thu Feb 24 | 7 | Hashing III : Open Addressing | 7 | Universal Hashing, Perfect Hashing |
| Sorting | PS3. Out: 3/1 | 5 | Tue Mar 01 | 8 | Sorting I : Insertion & Merge Sort, Master Theorem | 8 | Proof of Master Theorem, Examples |
| | Due: Mon 3/7 | | Thu Mar 03 | 9 | Sorting II : Heaps | 9 | Heap Operations |
| | HW lab: Sun 3/6 | 6 | Tue Mar 08 | 10 | Sorting III: Lower Bounds, Counting Sort, Radix Sort | 10 | Models of computation |
| | | | Wed Mar 09 | Q1 | Quiz 1 in class at 7:30pm. Covers L1-R10. Review Session on Tue 3/8 at 7:30pm. | | |
| Graphs | PS4. Out: 3/10 | | Thu Mar 10 | 11 | Searching I: Graph Representation, Depth-1st Search | 11 | Strongly connected components |
| and | Due: Fri 3/18 | 7 | Tue Mar 15 | 12 | Searching II: Breadth-1st Search, Topological Sort | 12 | Rubik's Cube Solving |
| Search | HW lab:W 3/16 | | Thu Mar 17 | 13 | Searching III: Games, Network properties, Motifs | 13 | Subgraph isomorphism |
| Shortest | PS5 | 8 | Tue Mar 29 | 14 | Shortest Paths I: Introduction, Bellman-Ford | 14 | Relaxation algorithms |
| Paths | Out: 3/29 | | Thu Mar 31 | 15 | Shortest Paths II: Bellman-Ford, DAGs | 15 | Shortest |
| | Due: Mon 4/11 | 9 | Tue Apr 05 | 16 | Shortest Paths III: Dijkstra | 16 | Speeding |
| | HW lab:Sun 4/10 | | Thu Apr 07 | 17 | Graph applications, Genome Assembly | 17 | Euler To |
| Dynamic | PS6 | 10 | Tue Apr 12 | 18 | DP I: Memoization, Fibonacci, Crazy Eights | 18 | Limits of dynamic programming |
| Program | Out: Tue 4/12 | | Wed Apr 13 | Q2 | Quiz 2 in class at 7:30pm. Covers L11-R17. Review Session on Tue 4/13 at 7:30pm. | | |
| ming | Due: Fri 4/29 | | Thu Apr 14 | 19 | DP II: Shortest Paths, Genome sequence alignment | 19 | Edit Distance, LCS, cost functions |
| | HW lab:W 4/27 | 11 | Tue Apr 19 | - | Patriot's Day - Monday and Tuesday Off | - | No recitation |
| | | | Thu Apr 21 | 20 | DP III: Text Justification, Knapsack | 20 | Saving Princess Peach |
| | | 12 | Tue Apr 26 | 21 | DP IV: Piano Fingering, Vertex Cover, Structured DP | 21 | Phylogeny |
| Numbers | PS7 out Thu4/28 | | Thu Apr 28 | 22 | Numerics I - Computing on large numbers | 22 | Models of computation return! |
| Pictures | Due: Fri 5/6 | 13 | Tue May 3 | 23 | Numerics II - Iterative algorithms, Newton's method | 23 | Computing the nth digit of π |
| (NP) | HW lab: Wed 5/4 | | Thu May 5 | 24 | Geometry: Line sweep, Convex Hull | 24 | Closest pair |
| | | 14 | Tue May 10 | 25 | Complexity classes, and reductions | 25 | Undecidability of Life |
| Beyond | | | Thu May 12 | 26 | Research Directions (15 mins each) + related classes | | |
| | | 15 | Finals week | Q3 | Final exam is cumulative L1-L26. Emphasis on L18-L26. Review Session on Fri 5/13 at 3pm | | |

**Dynamic Programming**

# Dynamic Programming

- Optimization technique, widely applicable
  - ➢Optimal substructure ➢Overlapping subproblems
- Today: Simple examples, alignment
  - Simple examples: Fibonacci, Crazy Eights
  - Alignment: Edit distance, molecular evolution
- Thursday: More DP
  - Alignment: Bound, Linear Space, Affine Gaps
  - Back to paths: All Pairs Shortest Paths DP1,DP2
- Next week:
  - Knapsack (shopping cart) problem
  - Text Justification
  - Structured DP: Vertex Cover on trees, phylogeny

# Today: Dynamic programming

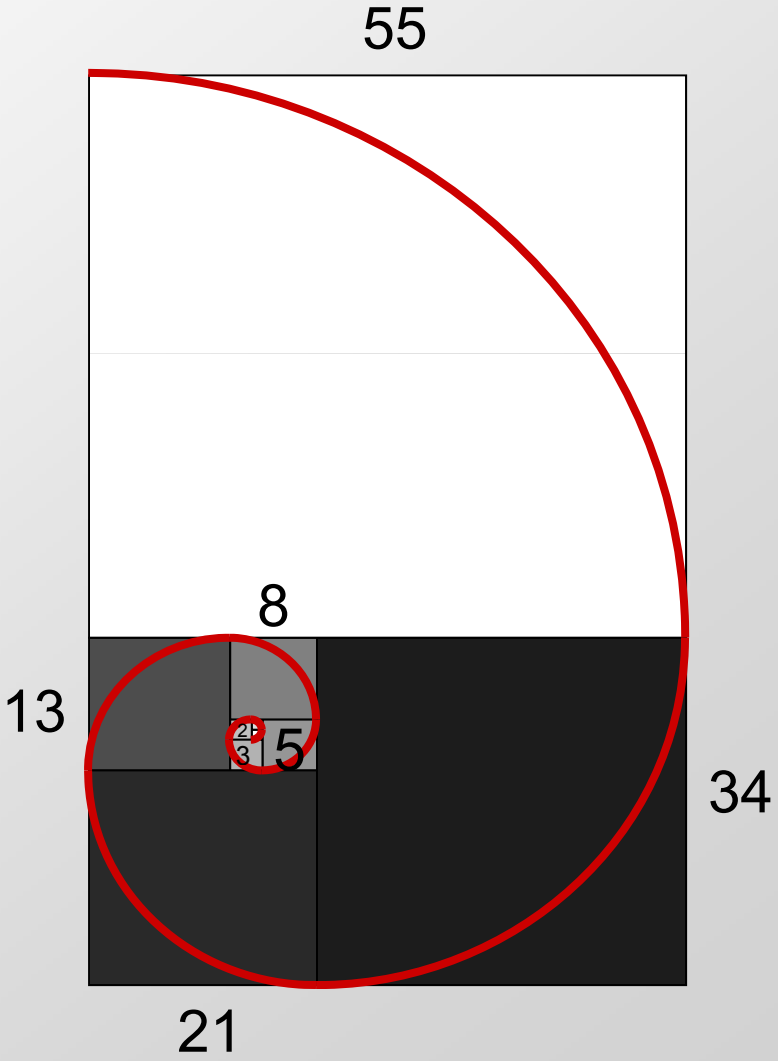
- Fibonacci numbers
  - Top-down vs. bottom-up
- Principles of Dynamic programming
  - Optimal sub-structure, repeated subproblems
- Crazy Eights
  - One-dimensional optimization
- Sequence alignment
  - Two-dimensional optimization

# 1. Fibonacci Computation

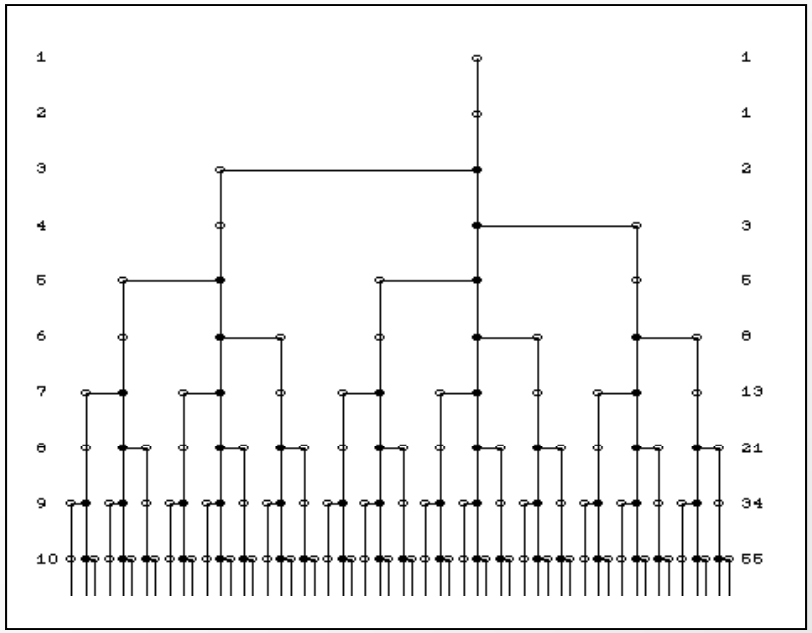
(not really an optimization problem,
but similar intuition applies)

# A simple introduction to Dynamic Programming

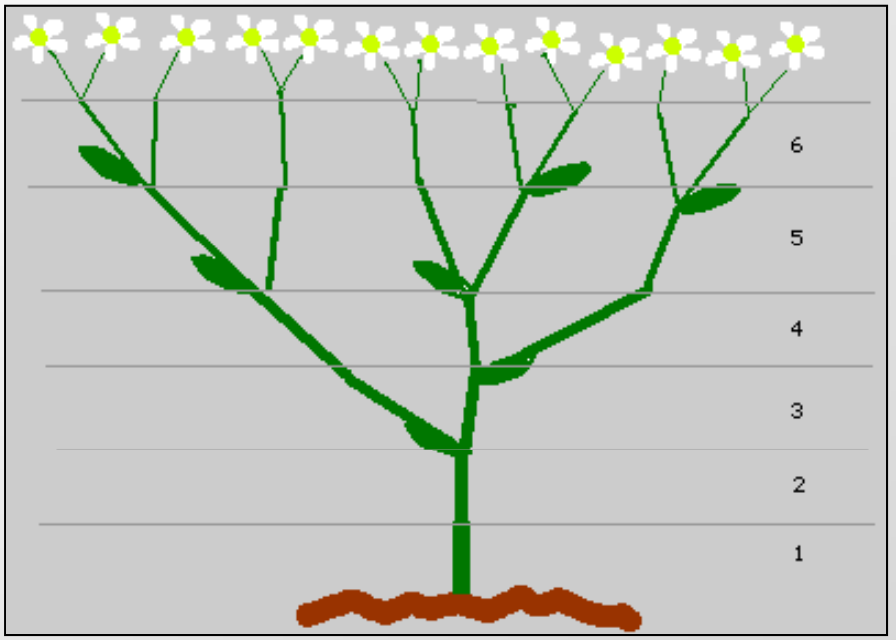
- Fibonacci numbers



# Fibonacci numbers are ubiquitous in nature



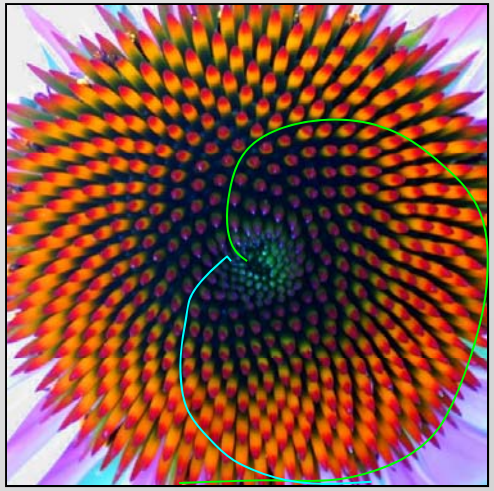

Rabbits per generation




Leaves per height

Romanesque spirals

Nautilus size



Coneflower spirals

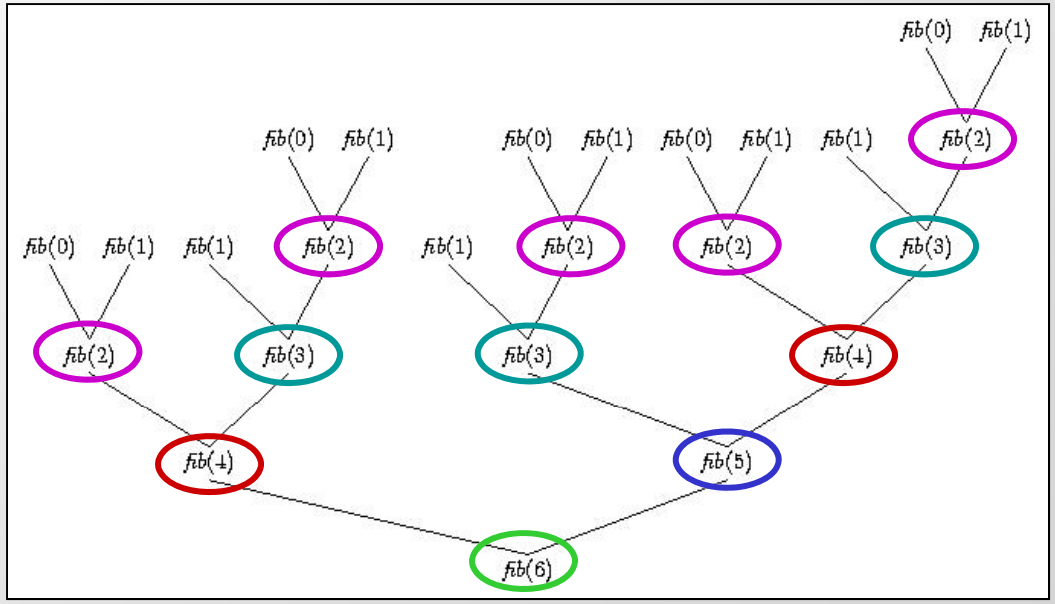
Leaf ordering

# Computing Fibonacci numbers: Top down

- Fibonacci numbers are defined recursively:
  - Python code

```
def fibonacci(n):
    if n==1 or n==2: return 1
    return fibonacci(n-1) + fibonacci(n-2)
```

- Goal: Compute $n^{th}$ Fibonacci number.
  - F(0)=1, F(1)=1, F(n)=F(n-1)+F(n-2)
  - 1,1,2,3,5,8,13,21,34,55,89,144,233,377,…
- Analysis:
  - T(n) = T(n-1) + T(n-2) = (…) = $O(2^n)$



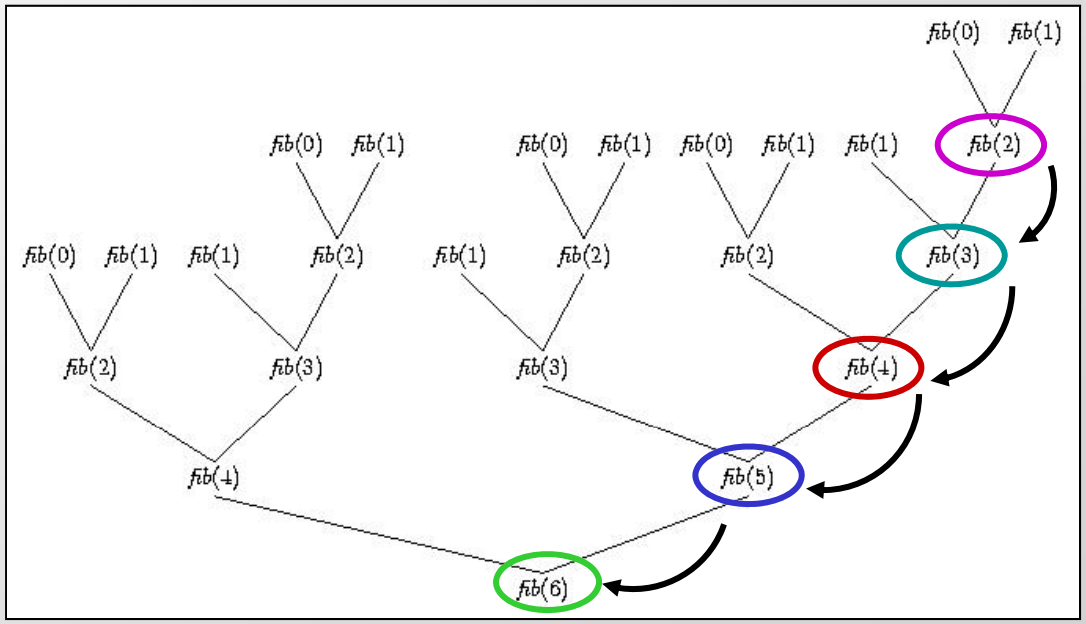
# Computing Fibonacci numbers: Bottom up

- Top-down approach
  - Python code

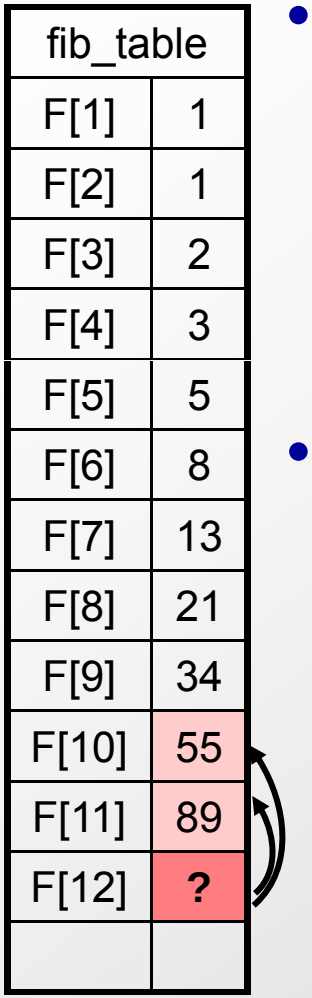
| fib_table | |
|---|---|
| F[1] | 1 |
| F[2] | 1 |
| F[3] | 2 |
| F[4] | 3 |
| F[5] | 5 |
| F[6] | 8 |
| F[7] | 13 |
| F[8] | 21 |
| F[9] | 34 |
| F[10] | 55 |
| F[11] | 89 |
| F[12] | ? |
| | |

```
def fibonacci(n):
        fib_table[1] = 1
        fib_table[2] = 1
        for i in range(3,n+1):
          fib_table[i] = fib_table[i-1]+fib_table[i-2]
        return fib_table[n]
```

  - Analysis: T(n) = O(n)



# Lessons from iterative Fibonacci algorithm




- What did the iterative solution do?
  - Reveal identical sub-problems
  - Order computation to enable result reuse
  - Systematically filled-in table of results
  - Expressed larger problems from their subparts
- Ordering of computations matters
  - Naïve top-down approach very slow
    - results of smaller problems not available
    - repeated work
  - Systematic bottom-up approach successful
    - Systematically solve each sub-problem
    - Fill-in table of sub-problem results in order.
    - Look up solutions instead of recomputing

# Dynamic Programming in Theory

- Hallmarks of Dynamic Programming
  - **Optimal substructure:** Optimal solution to problem (instance) contains optimal solutions to sub-problems
  - **Overlapping subproblems:** Limited number of distinct subproblems, repeated many many times
- Typically for optimization problems (unlike Fib example)
  - Optimal choice made locally: max( subsolution score)
  - Score is typically added through the search space
  - Traceback common, find optimal path from indiv. choices
- Middle of the road in range of difficulty
  - Easier: greedy choice possible at each step
  - DynProg: requires a traceback to find that optimal path
  - Harder: no opt. substr., e.g. subproblem dependencies

# Hallmarks of optimization problems

| Greedy algorithms | Dynamic Programming |
| --- | --- |
| ***1. Optimal substructure***<br>*An optimal solution to a problem (instance) contains optimal solutions to subproblems.* | |
| ***2. Overlapping subproblems***<br>*A recursive solution contains a "small" number of distinct subproblems repeated many times.* | |
| ***3. Greedy choice property***<br>*Locally optimal choices lead to globally optimal solution* | ***Greedy Choice is not possible***<br>*Globally optimal solution requires trace back through many choices* |

# Dynamic Programming in Practice

- Setting up dynamic programming
  1. Find ‘matrix’ parameterization (# dimensions, variables)
  2. Make sure sub-problem space is finite! (not exponential)
     - If not all subproblems are used, better off using memoization
     - If reuse not extensive, perhaps DynProg is not right solution!
  3. Traversal order: sub-results ready when you need them
     - Computation order matters! (bottom-up, but not always obvious)
  4. Recursion formula: larger problems = F(subparts)
  5. Remember choices: typically F() includes min() or max()
     - Need representation for storing pointers, is this polynomial !
- Then start computing
  1. Systematically fill in table of results, find optimal score
  2. Trace-back from optimal score, find optimal solution

# 2. Crazy Eights

One-dimensional Optimization

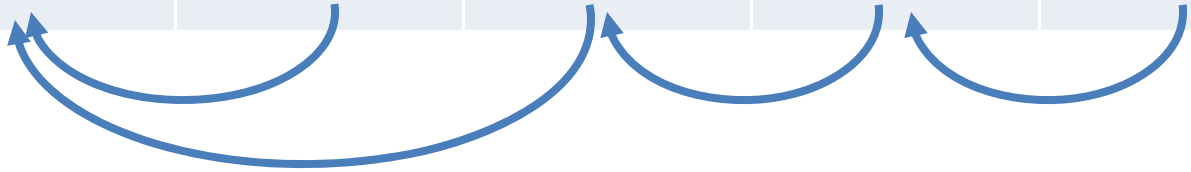
# Crazy 8s

- **Input:** a sequence of cards c[0]…c[n-1].
  - E.g., 7♣ 7♥ K♣ K♠ 8♥ …
- **Goal:** find the longest "trick subsequence" c[$i_1$]…c[$i_k$], where $i_1 < i_2 <\ldots< i_k$.
- For it to be a trick subsequence, it must be that:

  ∀ $j$, c[$i_j$] and c[$i_{j+1}$] "match" i.e.
  - they either have the same rank,
  - or the same suit
  - or one of them is an 8
  - in this case, we write: c[$i_j$] ~ c[$i_{j+1}$]
  - E.g., 7♣ K♣ K♠ 8♥ is the longest such subsequence in the above example

# Crazy8: Example computation

| | i=1 | i=2 | i=3 | i=4 | i=5 |
|---|---|---|---|---|---|
| c[i] | 7♣ | 7♥ | K♣ | K♠ | 8♥ |
| max score[i] | 1 | 2 | 2 | 3 | 4 |



**Rules:**

- **same rank**
- **or same suit**
- **or one is an 8**

# Dynamic Programming for Crazy Eights

- Setting up dynamic programming
  1. Find ‘matrix’ parameterization
     - One-dimensional array
  2. Make sure sub-problem space is finite! (not exponential)
     - Indeed, just one-dimensional array
  3. Traversal order: sub-results ready when you need them
     - Left-to-right ensures this
  4. Recursion formula:  larger problems = F(subparts)
     - Scan entire sub-array completed so far O(n) each step
  5. Remember choices: typically F() includes min() or max()
     - Pointer back to the entry that gave us optimal choice
- Then start computing
  1. Systematically fill in table of results, find optimal score
  2. Trace-back from optimal score, find optimal solution

# Crazy8: Max Score Algorithm

- Let trick($i$) be the length of the longest trick subsequence that ends at card c[$i$]
- **Question:** How can I relate value of trick($i$) with the values of trick($1$),…,trick($i$-$1$)?
- Recursive formula:

$$\text{trick}(i) = 1 + \max_{j<i,\ c[i] \sim c[j]} \text{trick}(j)$$

- Maximum trick length:

$$\max_i \text{trick}(i)$$

# Implementations

## Recursive

- memo = { }
- trick($i$):
  - if $i$ in memo: return memo[$i$]
  - else if $i$=1: return 1
  - else
    - $f$ := 1+$\max_{j<i,\ c[i] \sim c[j]}$ trick($j$)
    - memo[$i$] := $f$
    - return $f$
- call trick($n$)
- return maximum value in memo

# Implementations (cont.)

## Iterative

memo = { }

for *i=1* to *n*

    memo[*i*]= 1+$\max_{j<i,\ c[i] \sim c[j]}$ memo[*j*]

return maximum value in memo

Runtime: $O(n^2)$

# Dynamic Programming

- DP ≈ Recursion + Memoization
- DP works when:

**Optimal substructure**

An solution to a problem can be obtained by solutions to subproblems.

$$\text{trick}(i) = 1 + \max_{j>i,\ c[i] \sim c[j]} \text{trick}(j)$$

moreover….

**Overlapping Subproblems**

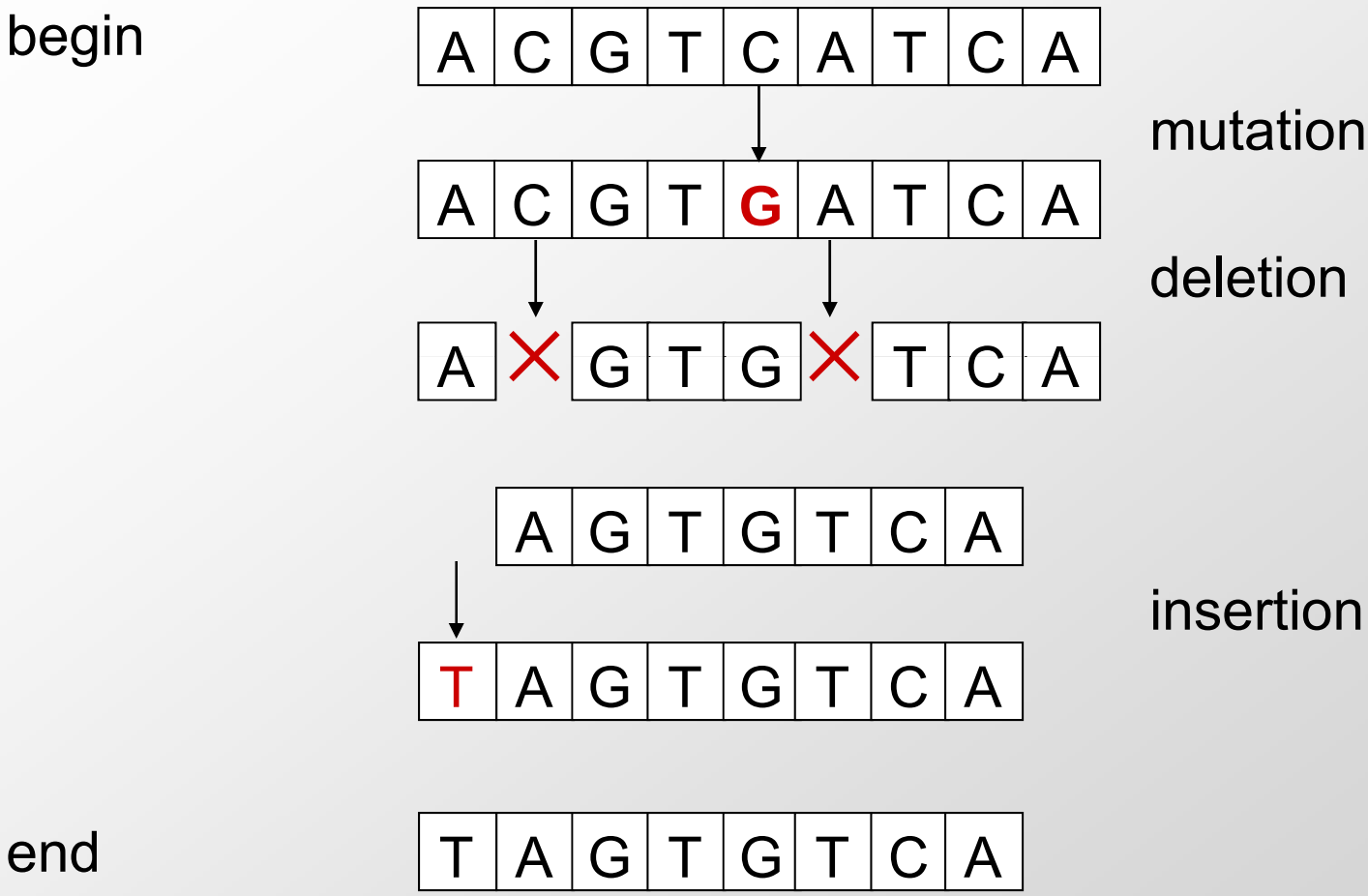
A recursive solution contains a "small" number of distinct subproblems (repeated many times)

trick(0), trick(1),…, trick($n$-1)

# 3. Sequence Alignment

Two-dimensional optimization

# Genomes change over time



# Goal of alignment: Infer edit operations

begin

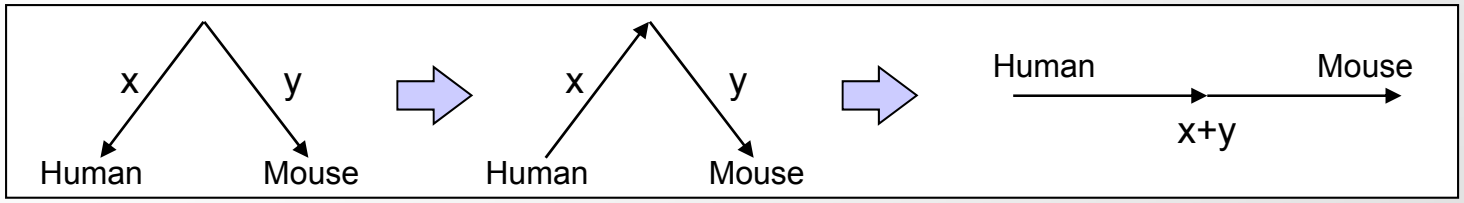
| A | C | G | T | C | A | T | C | A |
|---|---|---|---|---|---|---|---|---|


end

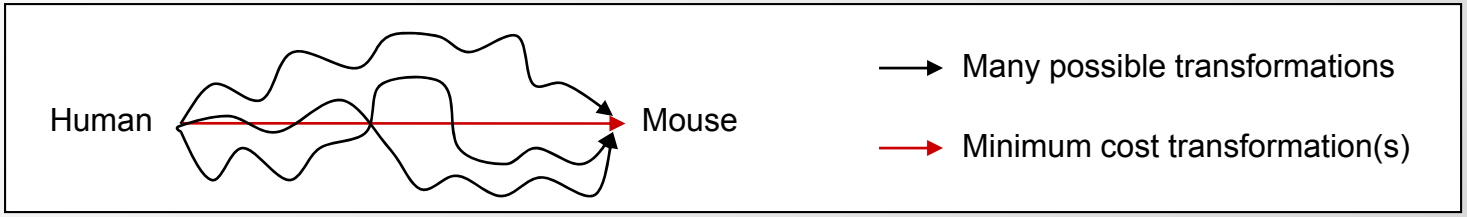
| T | A | G | T | G | T | C | A |
|---|---|---|---|---|---|---|---|

# From Bio to CS: Formalizing the problem

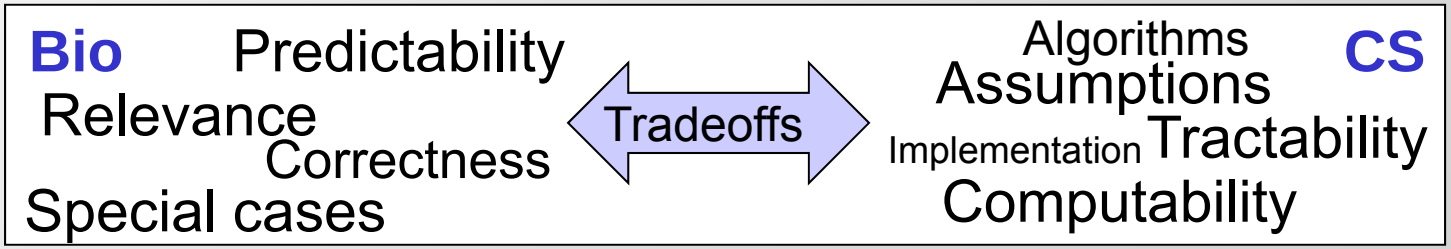
- Define set of evolutionary operations (insertion, deletion, mutation)
  - Symmetric operations allow time reversibility (part of design choice)




- Define optimality criterion (min number, min cost)
  - Impossible to infer exact series of operations (Occam's razor: find min)



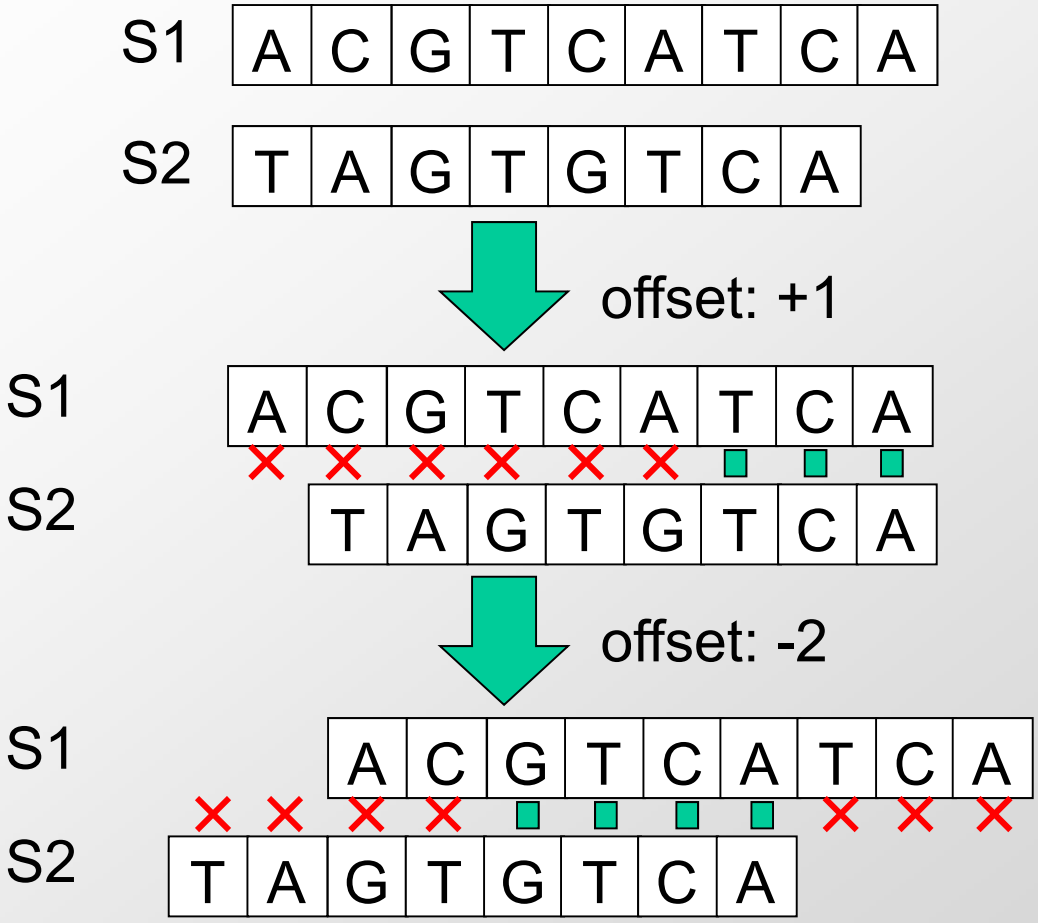

- Design algorithm that achieves that optimality (or approximates it)
  - Tractability of solution depends on assumptions in the formulation




Note: Not all decisions are conflicting (some are both relevant and tractable)
(e.g. Pevzner vs. Sankoff and directionality in chromosomal inversions)

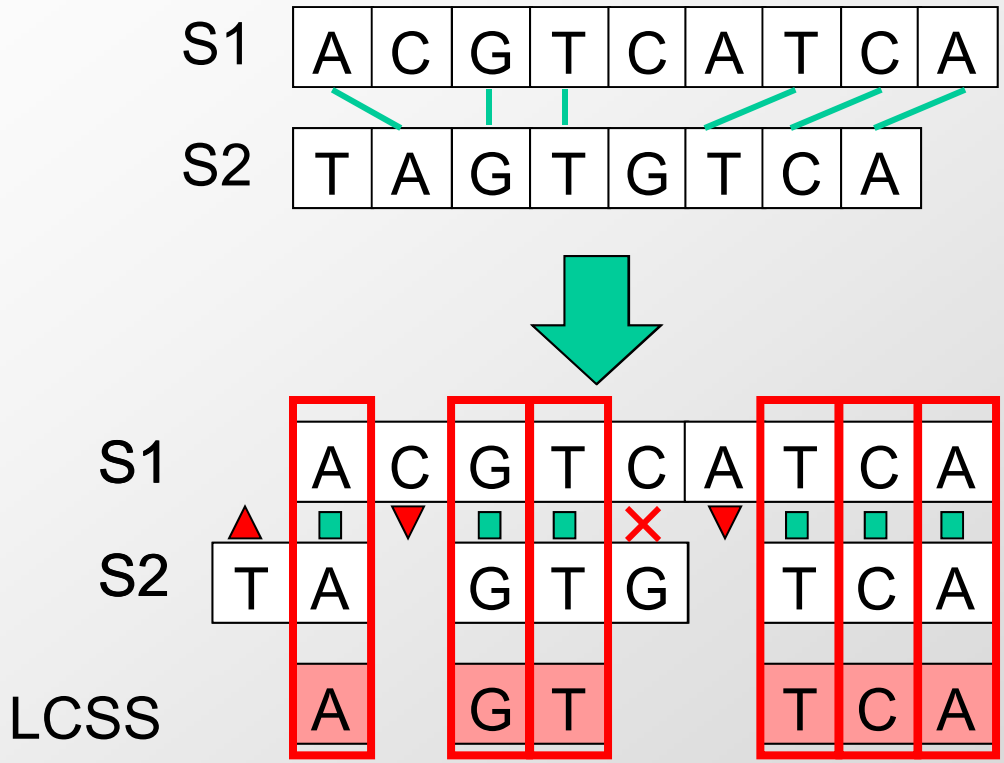
# Formulation 1: Longest common substring

- Given two possibly related strings S1 and S2
  - What is the longest common substring? (no gaps)



# Formulation 2: Longest common subsequence

- Given two possibly related strings S1 and S2
  - What is the longest common subsequence? (gaps allowed)




Edit distance:

- Number of changes needed for S1→S2
- Uniform scoring function

# Formulation 3: Sequence alignment

- Allow gaps (fixed penalty)
  - Insertion & deletion operations
  - Unit cost for each character inserted or deleted
- Varying penalties for edit operations
  - Transitions (Pyrimidine⇔Pyrimidine, Purine⇔Purine)
  - Transversions (Purine ⇔ Pyrimidine changes)
  - Polymerase confuses Aw/G and Cw/T more often

Scoring function:
Match(x,x) = +1
Mismatch(A,G)= -½
Mismatch(C,T)= -½
Mismatch(x,y) = -1

| | A | G | T | C |
|---|---|---|---|---|
| A | +1 | -½ | -1 | -1 |
| G | -½ | +1 | -1 | -1 |
| T | -1 | -1 | +1 | -½ |
| C | -1 | -1 | -½ | +1 |

purine pyrimid.

**Transitions**:
A⇔G, C⇔T common (lower penalty)

**Transversions**:
All other operations

# How can we compute the optimal alignment

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| S1 | A | C | G | T | C | A | T | C | A |
| S2 | T | A | G | T | G | T | C | A | |

- Given additive scoring function:
  - Cost of mutation (AG, CT, other)
  - Cost of insertion / deletion
  - Reward of match
- Need algorithm for inferring best alignment
  - Enumeration?
  - How would you do it?
  - How many alignments are there?

# Can we simply enumerate all possible alignments?

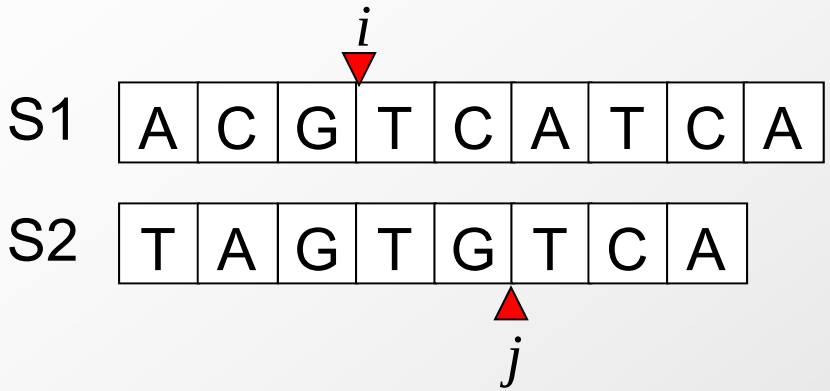
- Ways to align two sequences of length m, n

$$\binom{n+m}{m} = \frac{(m+n)!}{(m!)^2} \approx \frac{2^{m+n}}{\sqrt{\pi \cdot m}}$$

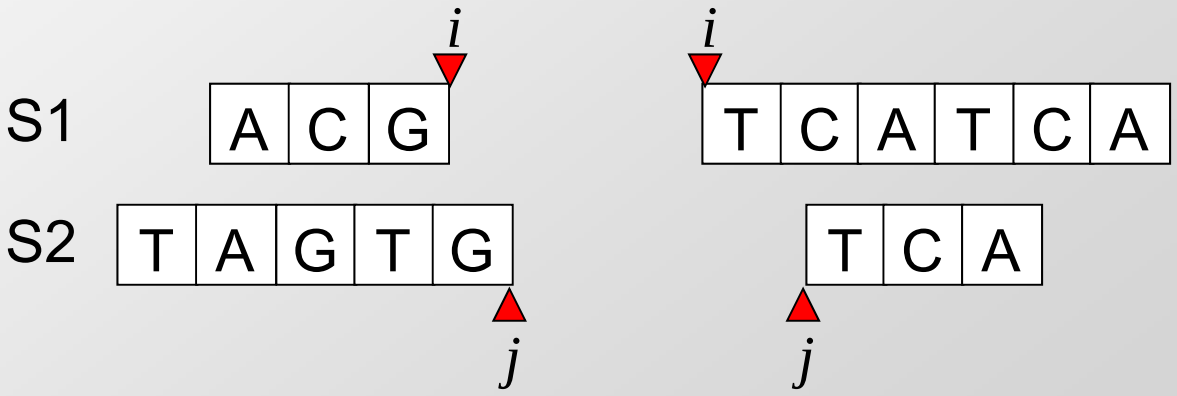
- For two sequences of length n

| n | Enumeration | Today's lecture |
|---|---|---|
| 10 | 184,756 | 100 |
| 20 | 1.40E+11 | 400 |
| 100 | 9.00E+58 | 10,000 |

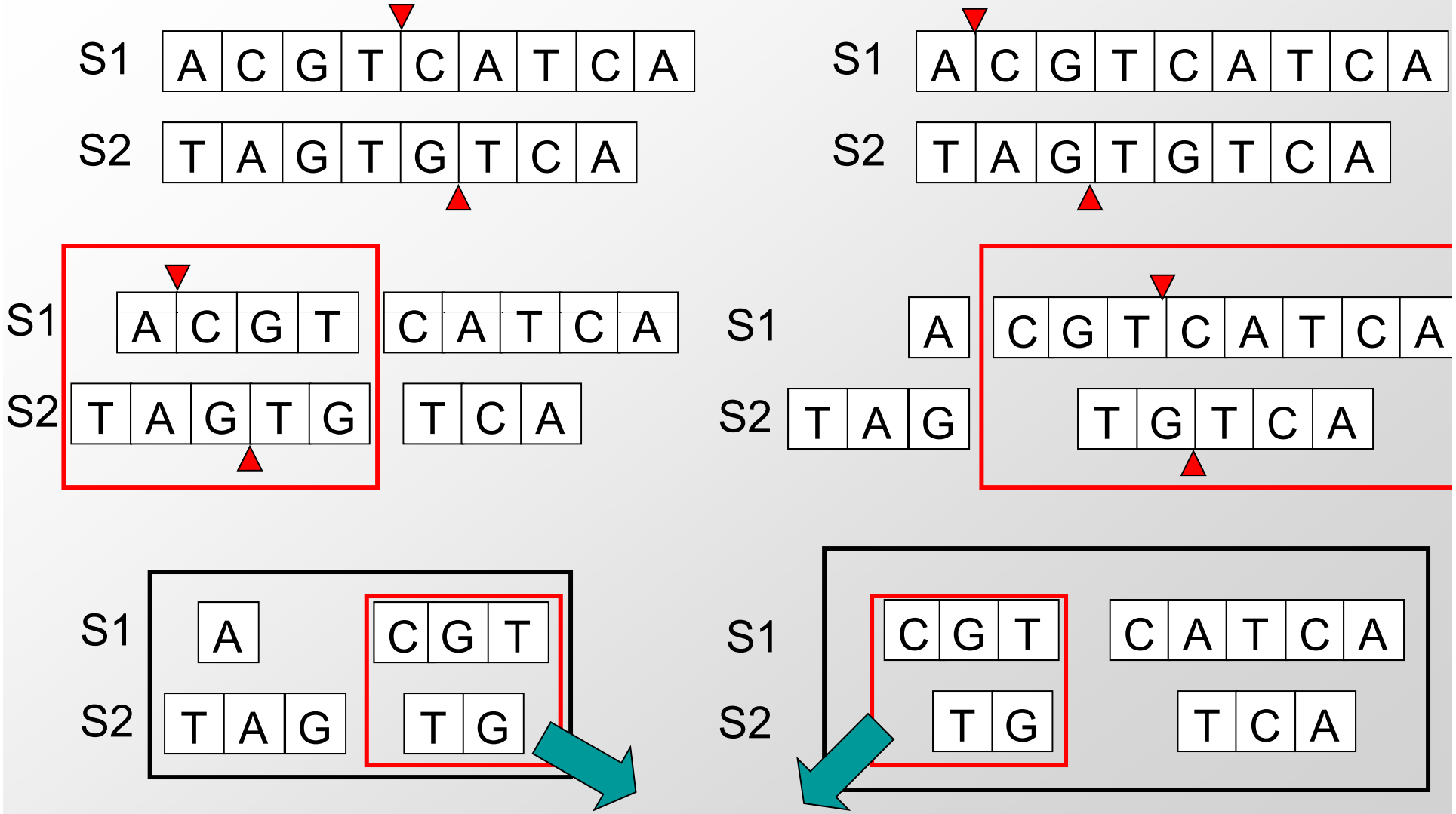
# Key insight: score is additive!




- Compute best alignment recursively
  - For a given aligned pair ***(i, j)***, the best alignment is:
    - Best alignment of S1[1..i] and S2[1..j]
    - \+ Best alignment of S1[ i..n] and S2[ j..m]



# Key insight: re-use computation




Identical sub-problems! We can reuse our work!

# Solution #1 – Memoization

- Create a big dictionary, indexed by aligned seqs
  - When you encounter a new pair of sequences
  - If it is in the dictionary:
    - Look up the solution
  - If it is not in the dictionary
    - Compute the solution
    - Insert the solution in the dictionary
- Ensures that there is no duplicated work
  - Only need to compute each sub-alignment once!

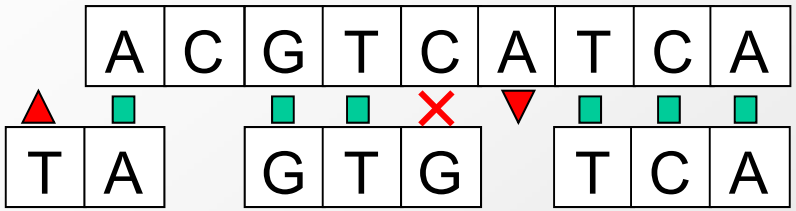
**Top down approach**

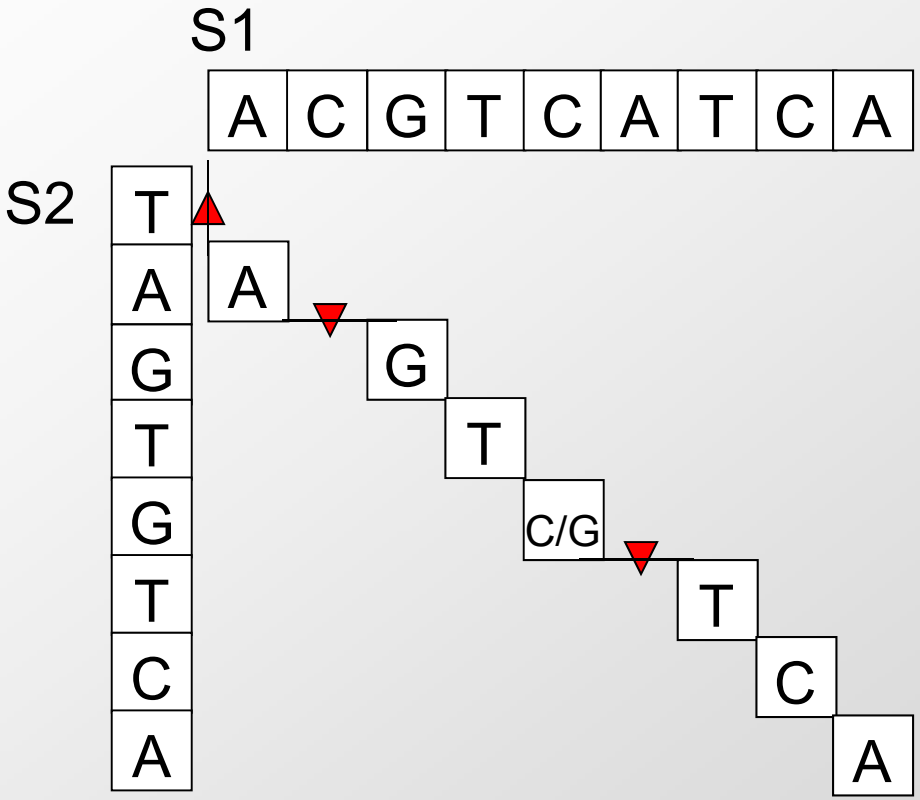
# Solution #2 – Dynamic programming

- Create a big table, indexed by (i,j)
  - Fill it in from the beginning all the way till the end
  - You know that you'll need every subpart
  - Guaranteed to explore entire search space
- Ensures that there is no duplicated work
  - Only need to compute each sub-alignment once!
- Very simple computationally!

**Bottom up approach**

# Duality: seq. alignment ⇔ path through the matrix

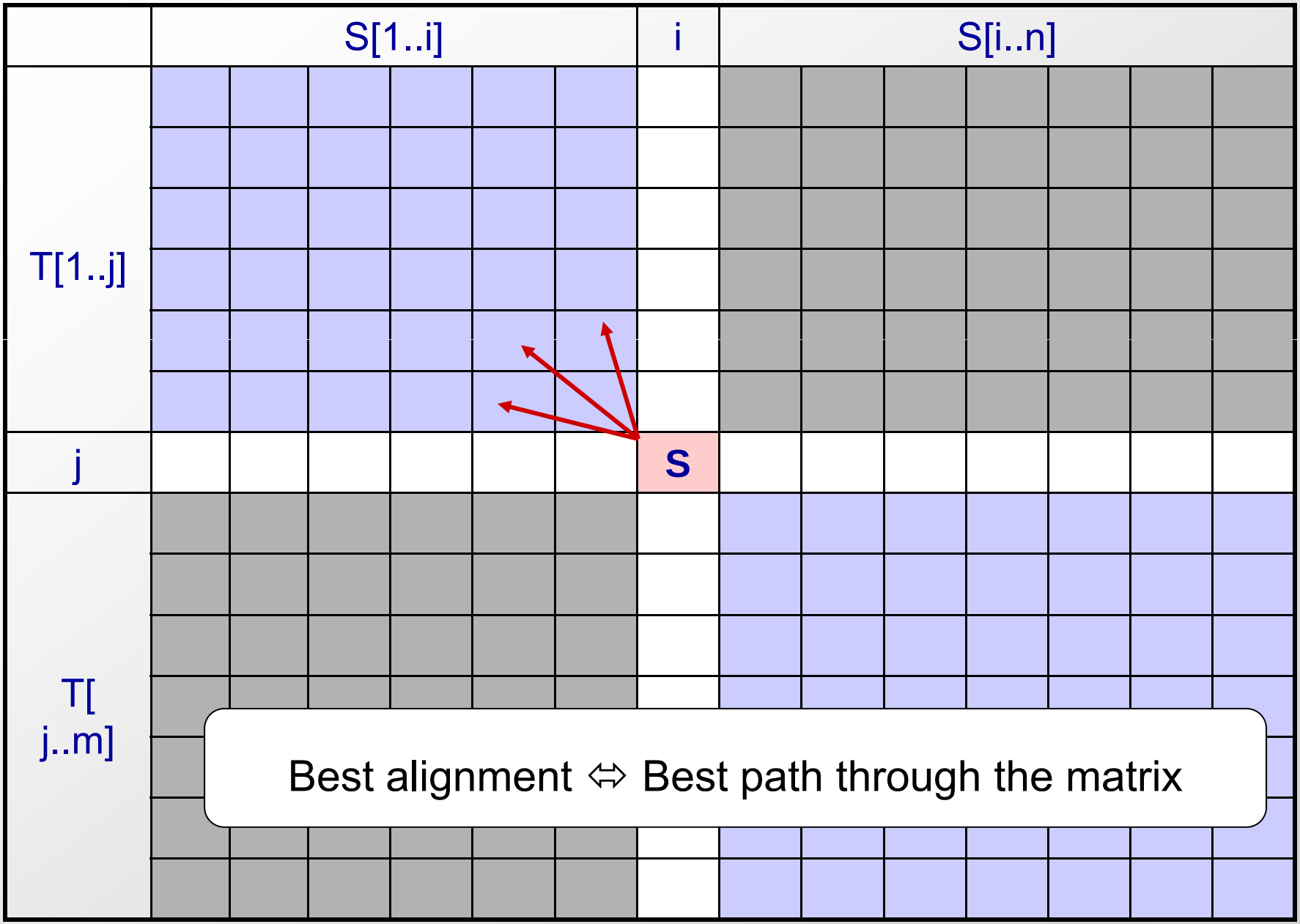


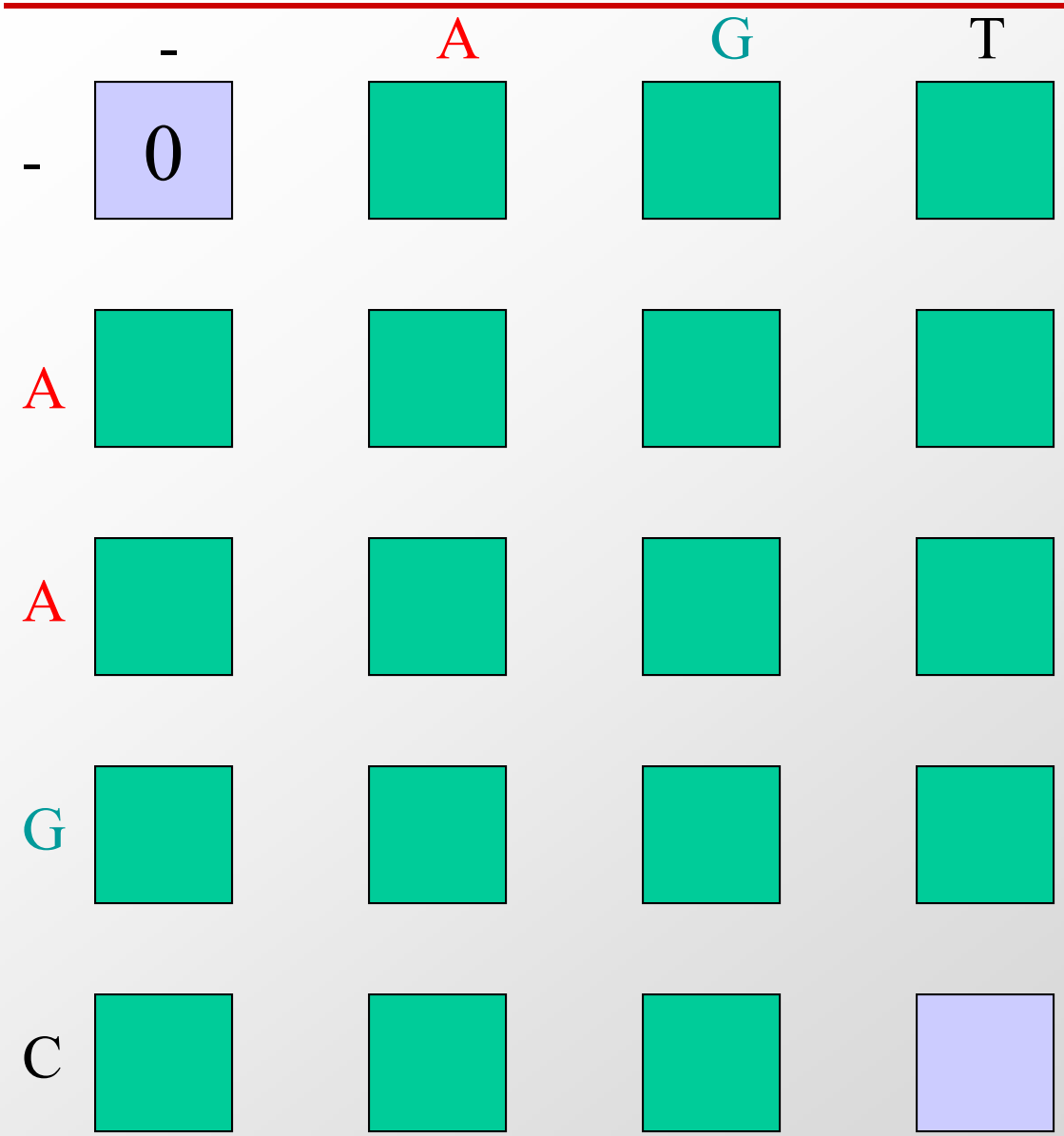

**Goal:**

**Find best path through the matrix**

# (1, 2, 3) Store score of aligning (i,j) in matrix M(i,j)



# Setting up the scoring matrix




**Initialization:**

- Top left: 0

**Update Rule:**

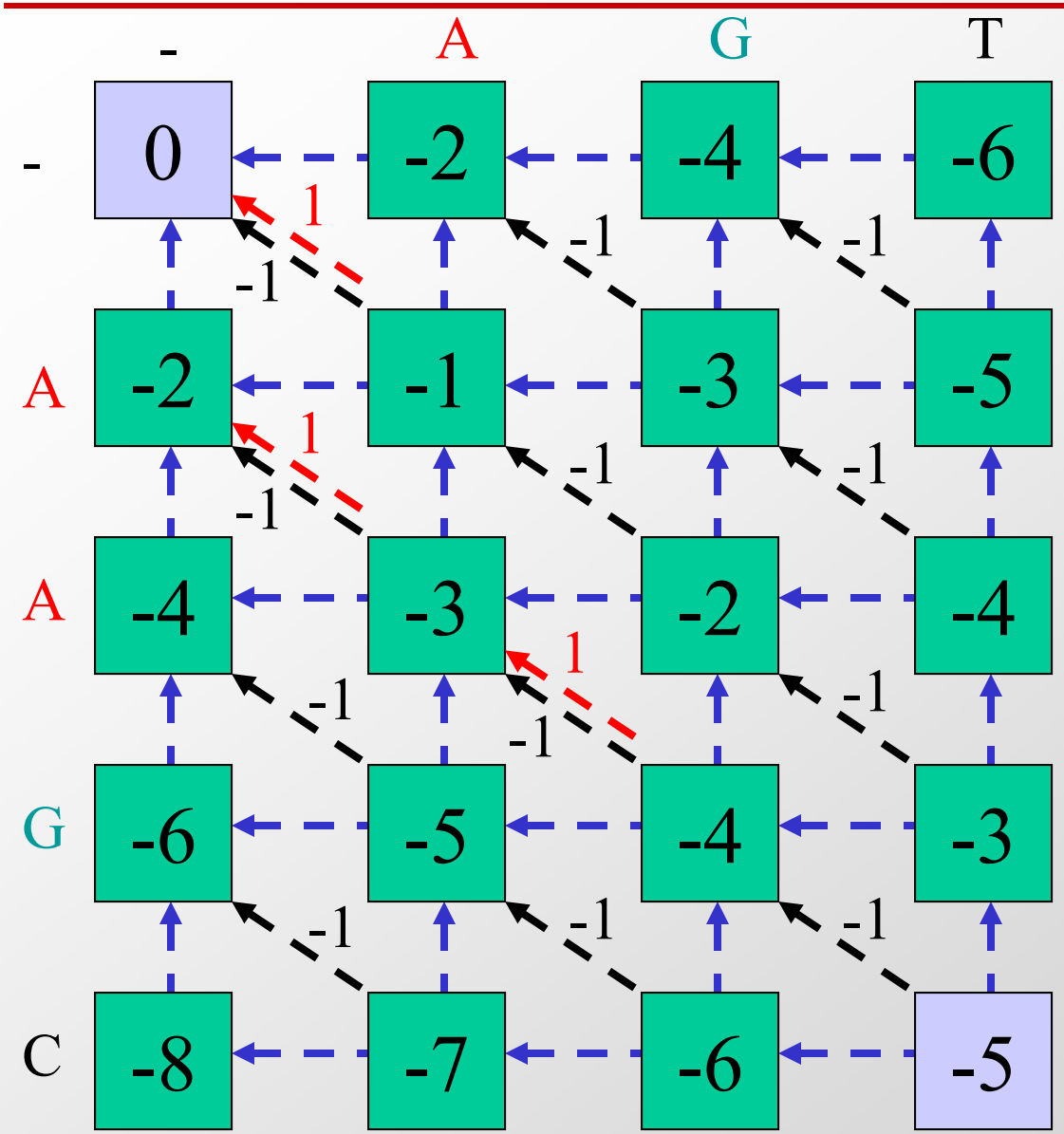
A(*i*,*j*)=max{

}

**Termination:**

- Bottom right

# Setting up graph of scores




**Initialization:**

- Top left: 0

**Update Rule:**

A(*i*,*j*)=max{

- A(*i*-1 , *j* ) - 2 gap
- A( *i* , *j*-1) - 2 gap
- A(*i*-1 , *j*-1) -1 mismatch
- A(*i*-1 , *j*-1)+1 match

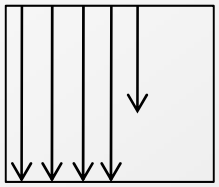
}

**Termination:**

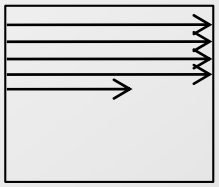
- Bottom right

# Dynamic Programming for sequence alignment

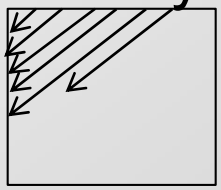
- Setting up dynamic programming
  1. Find 'matrix' parameterization
     - Prefix parameterization. Score(S[1..i],T[1..j]) ➔ F(i,j)
     - (i,j) only prefixes vs. (i,j,k,l) all substrings ➔ simpler 2-d matrix
  2. Make sure sub-problem space is finite! (not exponential)
     - It's just $n^2$, quadratic (which is polynomial, not exponential)
  3. Traversal order: sub-results ready when you need them



Cols
L→R



Rows
top→bot



Diags
topR→botL

  4. Recursion formula: larger problems = F(subparts)
     - Need formula for computing F(i,j) as function of previous results
     - Single increment at a time, only look at F(i-1,j), F(i,j-1), F(i-1,j-1) corresponding to 3 options: gap in S, gap in T, char in both
     - Score in each case depends on gap/match/mismatch penalties
  5. Remember choices: typically F() includes min() or max()
     - Remember which of three cells (top,left,diag) led to maximum

# More details on the recursion formula

- Computing the score of a cell from its neighbors

  $$F(i,j) = \max\begin{cases} F(i-1, j) - \text{gap} \\ F(i-1, j-1) + \text{score} \\ F(i, j-1) - \text{gap} \end{cases}$$

- Compute scores for prefixes of increasing length
  - This allows a single recursion (top-left to bottom-right) instead of two recursions (middle-to-outside top-down)
  - Only three possibilities for extending by one nucleotide: a gap in one species, a gap in the other, a (mis)match
  - When you reach bottom right, prefix of length $n$ is seq $S$
- Local update rules, only look at neighboring cells:
  - Compute next alignment based on previous alignment
  - Just like Fibonacci numbers: F[i] = F[i-1] + F[i-2]
  - Table lookup avoids repeated computation

# Today: Dynamic programming

- Fibonacci numbers
  - Top-down vs. bottom-up
- Principles of Dynamic programming
  - Optimal sub-structure, repeated subproblems
- Crazy Eights
  - One-dimensional optimization: *n* entries
  - Search all previous elements: $O(n^2)$ total time
- Sequence alignment
  - Two-dimensional optimization: $n^2$ entries
  - Search only 3 previous entries: $O(n^2)$ total time

# Dynamic Programming module

- Optimization technique, widely applicable
  - ➢Optimal substructure ➢Overlapping subproblems
- Today: Simple examples, alignment
  - Simple examples: Fibonacci, Crazy Eights
  - Alignment: Edit distance, molecular evolution
- Thursday: More DP
  - Alignment: Bound, Linear Space, Affine Gaps
  - Back to paths: All Pairs Shortest Paths DP1,DP2
- Next week:
  - Knapsack (shopping cart) problem
  - Text Justification
  - Structured DP: Vertex Cover on trees, phylogeny

# Course VI - 6.006 – Module VI – This is it

| Unit | Pset | Week | Date | | Lecture (Tuesdays and Thursdays) | | Recitation (Wed and Fri) |
|---|---|---|---|---|---|---|---|
| Intro | PS1 | 1 | Tue Feb 01 | 1 | Introduction and Document Distance | 1 | Python and Asymptotic Complexity |
| Binary | Out: 2/1 | | Thu Feb 03 | 2 | Peak Finding Problem | 2 | Peak Finding correctness & analysis |
| Search | Due: Mon 2/14 | 2 | Tue Feb 08 | 3 | Scheduling and Binary Search Trees | 3 | Binary Search Tree Operations |
| Trees | HW lab: Sun 2/13 | | Thu Feb 10 | 4 | Balanced Binary Search Trees | 4 | Rotations and AVL tree deletions |
| Hashing | PS2 Out: 2/15 | 3 | Tue Feb 15 | 5 | Hashing I : Chaining, Hash Functions | 5 | Hash recipes, collisions, Python dicts |
| | Due: Mon 2/28 | | Thu Feb 17 | 6 | Hashing II : Table Doubling, Rolling Hash | 6 | Probability review, Pattern matching |
| | HW lab:Sun 2/27 | 4 | Tue Feb 22 | - | President's Day - Monday Schedule - No Class | - | No recitation |
| | | | Thu Feb 24 | 7 | Hashing III : Open Addressing | 7 | Universal Hashing, Perfect Hashing |
| Sorting | PS3. Out: 3/1 | 5 | Tue Mar 01 | 8 | Sorting I : Insertion & Merge Sort, Master Theorem | 8 | Proof of Master Theorem, Examples |
| | Due: Mon 3/7 | | Thu Mar 03 | 9 | Sorting II : Heaps | 9 | Heap Operations |
| | HW lab: Sun 3/6 | 6 | Tue Mar 08 | 10 | Sorting III: Lower Bounds, Counting Sort, Radix Sort | 10 | Models of computation |
| | | | Wed Mar 09 | Q1 | Quiz 1 in class at 7:30pm. Covers L1-R10. Review Session on Tue 3/8 at 7:30pm. | | |
| Graphs | PS4. Out: 3/10 | | Thu Mar 10 | 11 | Searching I: Graph Representation, Depth-1st Search | 11 | Strongly connected components |
| and | Due: Fri 3/18 | 7 | Tue Mar 15 | 12 | Searching II: Breadth-1st Search, Topological Sort | 12 | Rubik's Cube Solving |
| Search | HW lab:W 3/16 | | Thu Mar 17 | 13 | Searching III: Games, Network properties, Motifs | 13 | Subgraph isomorphism |
| Shortest | PS5 | 8 | Tue Mar 29 | 14 | Shortest Paths I: Introduction, Bellman-Ford | 14 | Relaxation algorithms |
| Paths | Out: 3/29 | | Thu Mar 31 | 15 | Shortest Paths II: Bellman-Ford, DAGs | 15 | Shortest |
| | Due: Mon 4/11 | 9 | Tue Apr 05 | 16 | Shortest Paths III: Dijkstra | 16 | Speeding |
| | HW lab:Sun 4/10 | | Thu Apr 07 | 17 | Graph applications, Genome Assembly | 17 | Euler To |
| Dynamic | PS6 | 10 | Tue Apr 12 | 18 | DP I: Memoization, Fibonacci, Crazy Eights | 18 | Limits of dynamic programming |
| Program | Out: Tue 4/12 | | Wed Apr 13 | Q2 | Quiz 2 in class at 7:30pm. Covers L11-R17. Review Session on Tue 4/13 at 7:30pm. | | |
| ming | Due: Fri 4/29 | | Thu Apr 14 | 19 | DP II: Shortest Paths, Genome sequence alignment | 19 | Edit Distance, LCS, cost functions |
| | HW lab:W 4/27 | 11 | Tue Apr 19 | - | Patriot's Day - Monday and Tuesday Off | - | No recitation |
| | | | Thu Apr 21 | 20 | DP III: Text Justification, Knapsack | 20 | Saving Princess Peach |
| | | 12 | Tue Apr 26 | 21 | DP IV: Piano Fingering, Vertex Cover, Structured DP | 21 | Phylogeny |
| Numbers | PS7 out Thu4/28 | | Thu Apr 28 | 22 | Numerics I - Computing on large numbers | 22 | Models of computation return! |
| Pictures | Due: Fri 5/6 | 13 | Tue May 3 | 23 | Numerics II - Iterative algorithms, Newton's method | 23 | Computing the nth digit of π |
| (NP) | HW lab: Wed 5/4 | | Thu May 5 | 24 | Geometry: Line sweep, Convex Hull | 24 | Closest pair |
| | | 14 | Tue May 10 | 25 | Complexity classes, and reductions | 25 | Undecidability of Life |
| Beyond | | | Thu May 12 | 26 | Research Directions (15 mins each) + related classes | | |
| | | 15 | Finals week | Q3 | Final exam is cumulative L1-L26. Emphasis on L18-L26. Review Session on Fri 5/13 at 3pm | | |

**Dynamic Programming**