Sequence alignment

Brute force and dynamic programming algorithms

Outline

- The space of global alignments
- Dynamic programming algorithms

Pairwise Alignment: Task Definition

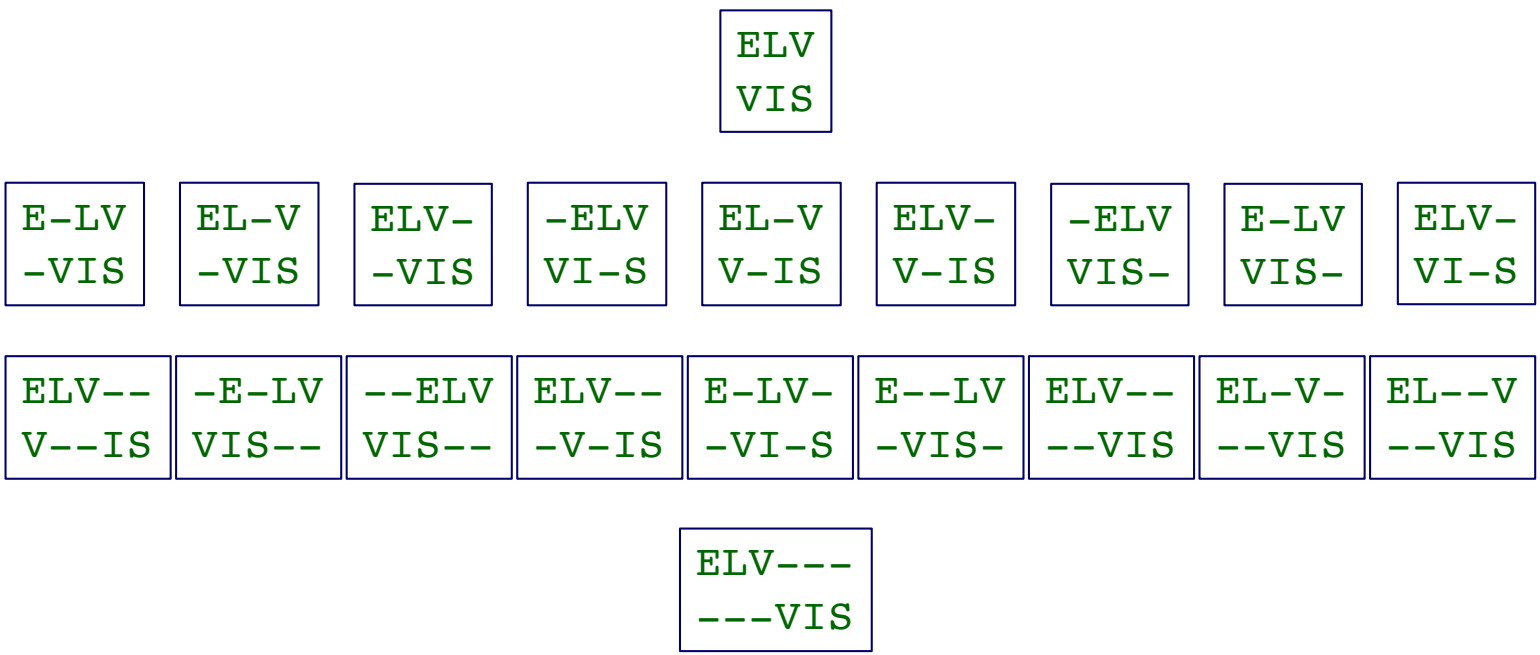
- Given
  - a pair of sequences (DNA or protein)
  - a method for scoring a candidate alignment
- Do
  - find an alignment for which the score is maximized

The brute force algorithm

```
best_alignment = null
for each possible alignment, a, of the sequences:
        if score(a) > score(best_alignment):
                best_alignment = a
return best_alignment
```

The Space of Global Alignments

the possible global alignments for ELV and VIS



Number of Possible Alignments

- given sequences of length m and n
- assume we don't count as distinct

```
C-      and   -C
-G            G-
```

- we can have as few as 0 and as many as $\min\{m, n\}$ aligned pairs
- therefore the number of possible alignments is given by

$$\sum_{k=0}^{\min\{m,n\}} \binom{n}{k}\binom{m}{k} = \binom{n+m}{n}$$

Number of Possible Alignments

- there are $\binom{2n}{n} = \frac{(2n)!}{(n!)^2} \approx \frac{2^{2n}}{\sqrt{\pi n}}$

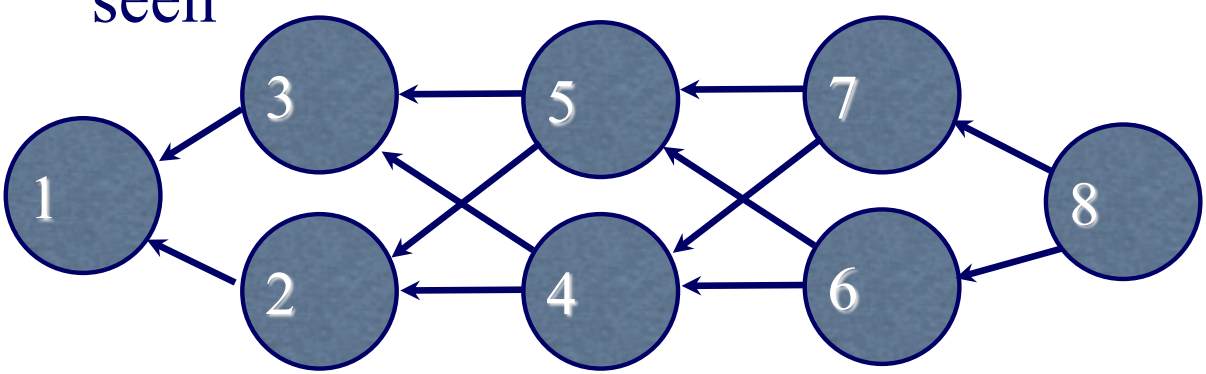
  possible global alignments for 2 sequences of length n
  - e.g. two sequences of length 100 have $\approx 10^{59}$ possible alignments
  - but we can use *dynamic programming* to find an optimal alignment efficiently

Dynamic Programming

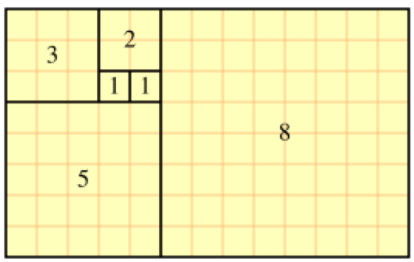
- Algorithmic technique for optimization problems that have two properties:
  - *Optimal substructure*: Optimal solution can be computed from optimal solutions to subproblems
  - *Overlapping subproblems*: Subproblems overlap such that the total number of distinct subproblems to be solved is relatively small

Dynamic Programming

- Break problem into overlapping subproblems
- use *memoization*: remember solutions to subproblems that we have already seen

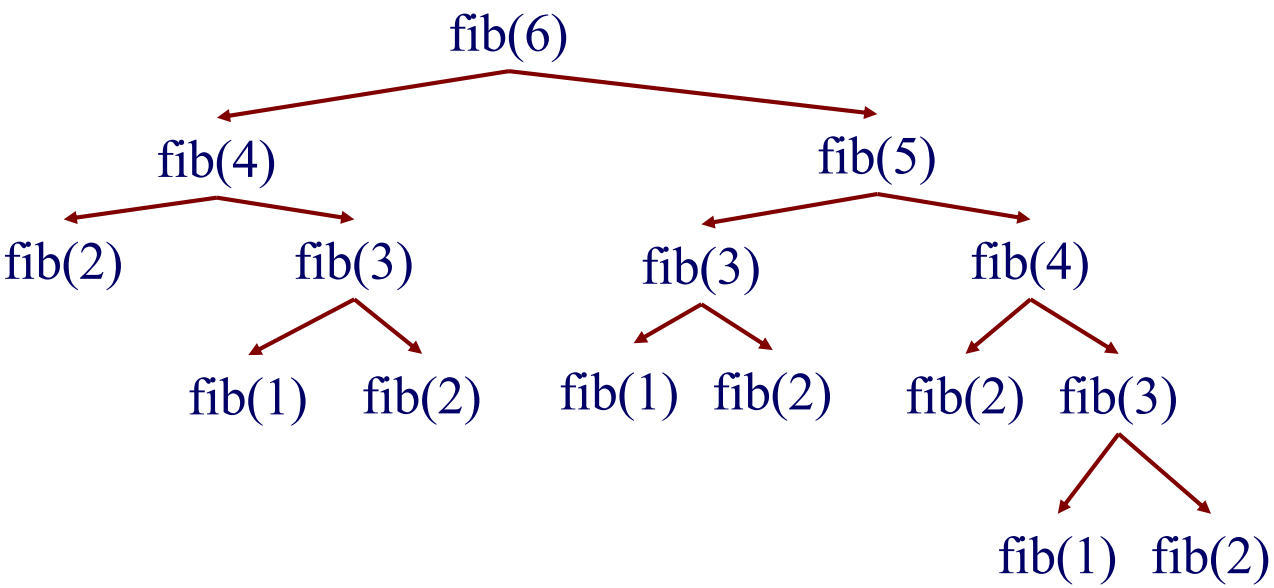


Fibonacci example



- 1,1,2,3,5,8,13,21,...
- *fib(n) = fib(n - 2) + fib(n - 1)*
- Could implement as a simple recursive function
- However, complexity of simple recursive function is exponential in *n*

Recursive Fibonacci call stack



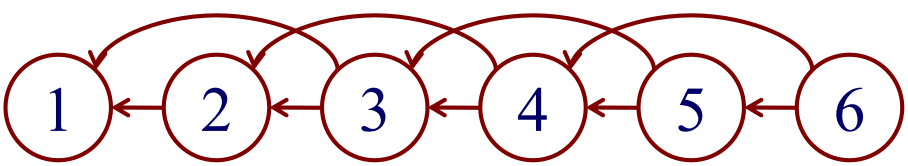
15 calls – only 6 unique calls

Fibonacci dynamic programming

- Two approaches
  1. *Memoization:* Store results from previous calls of function in a table (top down approach)
  2. Solve subproblems from smallest to largest, storing results in table (bottom up approach)
- Both require evaluating all (n-1) subproblems only once: O(n)

Dynamic Programming Graphs

- Dynamic programming algorithms can be represented by a directed acyclic graph
  - Each subproblem is a vertex
  - Direct dependencies between subproblems are edges



graph for fib(6)

Why “Dynamic Programming”?

- Coined by Richard Bellman in 1950 while working at RAND
- Government officials were overseeing RAND, disliked research and mathematics
- “programming”: planning, decision making (optimization)
- “dynamic”: multistage, time varying
- “It was something not even a Congressman could object to. So I used it as an umbrella for my activities”

Summary

- The space of possible alignments is huge!
- Brute force algorithms are infeasible
- Dynamic programming will offer a solution
- Two properties of dynamic programming problems
  - Optimal substructure
  - Overlapping subproblems