Dynamic Programming

Overview of dynamic programming

DP algorithm for pairwise alignment

References:
- Cormen, et al, Introduction to Algorithms [CIS 315 text]
- Giegerich & Wheeler, “Pairwise Sequence Alignment” (PDF on class web site)

Recursive Problems

- Self-similarity
  - subproblems are smaller instances of main problem
- Base case
  - smallest instance of the problem
- Recursive (inductive) case
  - how to solve large problem as combination of smaller ones
- Examples from math:
  - inductive reasoning
  - recurrence relations

Example Recursive Problem

- Tree and graph problems often have a natural recursive definition
  - tree = leaf(X) or node(L,R) where L, R are trees
- Algorithm to search for leaf A:
  - def search(T,A)
  - if T = leaf(X) /* base case? */
  - print “yes” if X = A
  - else /* T = tree(L,R) */
  - search(L) /* recurse left */
  - search(R) /* recurse right */
  - end
  - end

Efficiency of Recursive Algorithms

- In other problems, a top-down recursive algorithm is not a good choice
  - def fib(i)
    - if (i == 1 || i == 2)
      return 1
    - else
      return fib(i-1) + fib(i-2)
    - end
  - end
- O(2^n) steps to compute fib(n)
Efficiency (cont’d)

- What distinguishes the binary tree search from the evaluation of the Fibonacci function?
- Both involve binary trees...
- In the Fibonacci function the same value is computed more than once

<table>
<thead>
<tr>
<th></th>
<th>4</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

Note: two calls to fib(2) in the evaluation of fib(4)

Efficiency (cont’d)

- That wasn’t too bad, but the program gets out of hand for larger values of n

<table>
<thead>
<tr>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>0</td>
</tr>
</tbody>
</table>

Now fib(2) is called 5 times

Dynamic Programming

- Dynamic programming is a general technique for solving combinatorial optimization problems
- A problem is a candidate for solution with DP if:
  - it has a recursive formulation (e.g. defined by a recurrence relation)
  - it has overlapping subproblems (subproblems used in more than one problem)
  - it has optimal substructure: the optimal solution can be computed by finding the optimal solution of its subproblems
Dynamic Programming (cont’d)

- Dynamic programming involves
  - a recursive definition of the function to be optimized
  - recurrence for defining similar subproblems
  - base case
  - use of a data structure (“table”) to hold results of recursive calls
- Abstract: if a subproblem result is not in the table, evaluate, store result to be reused for the next call
- Practice: systematic (bottom-up) construction of table

Dynamic Programming (cont’d)

- To solve a problem with DP, locate the table cell that corresponds to the solution
- Fill in the values of the necessary subproblems
- Return the value of the solution cell
- Example: fib(7)
  - table = vector of n ≥ 7 elements
  - solution is the value of V[7]
  - value of fib(7) needs table entries for fib(6) and fib(5) so make recursive calls
  - eventually fills in table entries V[1]...V[7]
  - but note that fib(n) called only once for each n

Traceback

- For an optimization problem, the table entries give the value of the optimum
- To show how the solution was obtained, augment the table with traceback information
  - traceback = reference to subproblem(s) used to compute a cell entry
- A traceback path from the solution cell to the base case will produce the steps used to construct the solution

Traceback (cont’d)

- Example: text formatting
- The goal is to figure out where to place line breaks in order to minimize the “raggedness” of a paragraph
  - cost = function of amount of white space at ends of lines
  - V[i] = cost of paragraph if word i starts a line
  - bottom-up solution: compute V[n]..V[1]
  - traceback from V[1] shows where to place the line breaks...
Aside: Operations Research

- Where does the “program” in “dynamic programming” come from?
- In operations research (OR), a program is a table that represents a solution to an optimization problem
- Different types of programs involve differing sets of constraints on the table entries
  - linear programming
  - integer programming
  - ...

Pairwise Alignment

- From the previous lecture:
  - an alignment is a mapping between characters of two strings
  - each character in a string corresponds either to a character in the other string or to a space
    - S: AA-TACCG
    - T: AAACA-CG
  - The original strings do not have to be the same length
  - After a (global) alignment, the strings are the same length

Pairwise Alignment (cont’d)

- Each element in the mapping can be assigned a cost:
  - 0 if $S_i = T_i$ (match)
  - $m$ if $S_i \neq T_i$ (mismatch cost)
  - $g$ if $S_i \Rightarrow \cdot$ or $T_i \Rightarrow \cdot$ (gap cost)
- The cost of an alignment is the sum of the costs of each position
- Example:
  - S: AA-TACCG
  - T: AAACA-CG
  - cost = $m + 2g$

Pairwise Alignment (cont’d)

- A note about gaps:
  - Two space characters are never aligned with each other
  - A gap is a string of 1 or more consecutive space characters
  - Gap costs can be linear (as in the previous slide) or affine:
    - $p = g + n(e-1)$
    - where $g$ is a “gap opening” cost and $e$ is a “gap extension” cost
- For the rest of this lecture (and on the project) we’ll use linear gap penalties
Dot Matrix

- A dot matrix can be used to produce an alignment
  - label columns with chars from one string, rows with chars from the other string
  - enter a dot where row, column label are the same
  - A path from [1,1] to [n,m] defines an alignment
    diagonal = mis(match)
    down = delete
    right = insert

DP Algorithm for Pairwise Alignment

- The dot matrix is the basis for the recursive formulation of the pairwise alignment problem
- The dynamic programming table will be a matrix D
  - D[i,j] = minimum cost of aligning the first i chars of S (1 ≤ i ≤ n) and the first j chars of T (1 ≤ j ≤ m)
  - D[0,0] is the cost of aligning S1..i with 0 chars from j, i.e. inserting i spaces in front of T
  - D[0,j] defined similarly
  - D[i,0] and D[0,j] define the base cases (there is only one way to construct each alignment)

DP Alignment (cont’d)

- Base case:
  - D[0,0] = 0
  - D[i,0] = i*g for 1 ≤ i ≤ n
  - D[0,j] = j*g for 1 ≤ j ≤ m

- Recurrence: D[i,j] is the minimum of
  - D[i-1,j-1] (Si = Tj)
  - D[i-1,j-1] + m (Si ≠ Tj)
  - D[i-1,j] + g (delete, i.e. align Si with '-' in T)
  - D[i,j-1] + g (insert, i.e. align Tj with '-' in S)

Optimal Substructure

- For this formulation to work, we need to know D[i,j] can be decomposed into solutions of three adjoining cells
- Optimal substructure requirement of DP:
  The solution of D[i,j] must be a function of optimal solutions of its subproblems
Optimal Substructure (cont’d)

- Formal proof: see Gusfield, 1997
- Using the idea of an edit transcript, show $D[i,j]$ must be either
  - $D[i-1,j-1]$  
  - $D[i-1,j-1] + m$  
  - $D[i-1,j] + g$  
  - $D[i,j-1] + g$  
  (i.e. it is not necessary to look elsewhere in the table)

Method

- To align strings S and T:
  - allocate an array $D$ with $n+1$ rows and $m+1$ columns  
  - initialize the top row and left column using the values defined for the base case of the recurrence  
  - fill in the interior rows using the recurrence:

$$
D[i,j] = \min( (D[i-1,j]+g), (D[i,j-1]+g), (D[i-1,j-1]+c) )
$$

Alignment Traceback

- When the table is filled in, $D[n,m]$ will be the cost of the optimal alignment of the two strings  
- In order to know what alignment produces this minimal cost, record traceback information in each cell  
- The traceback at $D[i,j]$ indicates which of the three terms in the min function can be used to compute $D[i,j]$  
- Save more than one if there is a tie

Alignment Traceback (cont’d)

- Any traceback path from $D[n,m]$ to $D[0,0]$ will give the operations to construct an optimal alignment  
- Write the aligned strings from right to left  
- When in cell $D[i,j]$
  - write $S_i$ and $T_j$  
  - write $S_i$ and '-'  
  - write '-' and $T_j$
- Q: do all paths lead to $D[0,0]$? Why?
Example

- S = ATGTA
- T = ATTCA
- Matrix using
  - g = 1
  - m = 1
- What are the initial entries (base case)?

```
<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>T</th>
<th>T</th>
<th>C</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Example

- S = ATGTA
- T = ATTCA
- Matrix using
  - g = 1
  - m = 1
- What are the traceback entries for the base case?

```
<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>T</th>
<th>T</th>
<th>C</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>T</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>G</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>4</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Example

- S = ATGTA
- T = ATTCA
- Matrix using
  - g = 1
  - m = 1
- What is the value for D[1,1]?
- Its traceback entry?

```
<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>T</th>
<th>T</th>
<th>C</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>T</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>G</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>4</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Example

- S = ATGTA
- T = ATTCA
- Matrix using
  - g = 1
  - m = 1
- What are the remaining entries for row 1?

```
<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>T</th>
<th>T</th>
<th>C</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>T</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>G</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>4</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
Example

- **S** = ATGTA
- **T** = ATTCA
- Matrix using 
  - *g* = 1
  - *m* = 1
- Fill in the rest of the rows...

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>T</th>
<th>T</th>
<th>C</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>T</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>G</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>T</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>A</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

Note two tracebacks from D[1,5]

Example

- **S** = ATGTA
- **T** = ATTCA
- What is the cost of a complete alignment?
- How many such alignments are there?

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>T</th>
<th>T</th>
<th>C</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>T</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>G</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>T</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>A</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

Example

- Test your understanding:
  - is each value on a solution path less than or equal to the previous value?
  - is any sequence of adjacent cells with continually equal or decreasing values a solution?
  - i.e. (same question): why is there no arrow from D[5,5] to D[5,4]?
Example

- Test your understanding:
  - what would the matrix look like if there were no matches?
  
  e.g. align:
  
  AAAAA
  TTTTT
  
  what would the alignment cost be?
  
<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>T</th>
<th>T</th>
<th>C</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>T</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>G</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>T</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>A</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

Complexity

- This method is clearly $O(n \times m)$ in both space and time
  - Not suitable for very long sequences (e.g. two chromosomes…)
- It is possible to compute scores only in $O(n)$ space
  - the current row uses values from the previous row only
- There is a linear-space method that produces scores and alignments (Hirschberg, 1977)
- More on this and other improvements later in the term [?] 

Project #3: Pairwise Alignment

- The next programming project this term will be to write a pairwise alignment program
- Input: two sequences
  - recommend FASTA format, but any other “standard” format will do
- Output: optimal alignment, computed using a dynamic programming algorithm
- Runtime parameters:
  - mismatch cost
  - gap cost

Project #3 (cont’d)

- Some extra credit ideas:
  - affine gap penalties
  - end-gaps free
    - no cost for one or more spaces on the end of either string
  - character-specific mismatch penalties
    - e.g. $20 \times 20$ matrix of amino acid substitution costs
    - read matrix from a file when the program starts
Project #3 (cont’d)

- Project tar file will have:
  - C++ outline of main program
  - Array class for creating 2D matrices
designed for arrays of floating point numbers
  - May be retrofit to be template for arrays of any objects
  - Makefile
  - Test sequences in FASTA format