

LECTURE 6: DYNAMIC PROGRAMMING

STAT 545: INTRO. TO COMPUTATIONAL STATISTICS

Vinayak Rao

Purdue University

September 5, 2019

Solve a complex problem by breaking it into simpler problems

Recursion without recalculation:

- Relate solution of a problem to solutions of simpler problems (recursion)
- Identify and solve initial (base) problems
- Reuse existing solutions to compute more complicated solutions (memoization)

PROB 1: WHO IS THE TALLEST PERSON IN CLASS?

Setup: We can only compare heights one pair at a time.

Naïve approach: build a binary relation matrix:

$$\begin{array}{c} a \\ b \\ \vdots \\ N \end{array} \begin{array}{cccc} a & b & \dots & N \\ \left[\begin{array}{cccc} - & 1 & \dots & 1 \\ 0 & - & \dots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & - \end{array} \right] \end{array}$$

$O(N^2)$ comparisons, but lots of redundancy.

Can we do better? $O(N)$?

PROB 2: ORDER-DEPENDENT SUMS

Pick a set of unique integers. E.g. $\{1, 3, 4\}$.

Find the number of ways to write N as sums of these.

E.g. for $N = 5$, the answer is 6:

$$5 = 1 + 1 + 1 + 1 + 1$$

$$= 1 + 1 + 3$$

$$= 1 + 3 + 1$$

$$= 3 + 1 + 1$$

$$= 1 + 4$$

$$= 4 + 1$$

[http://web.stanford.edu/class/cs97si/
04-dynamic-programming.pdf](http://web.stanford.edu/class/cs97si/04-dynamic-programming.pdf)

ORDER-DEPENDENT SUMS (CONTD.)

How do we solve for $N = 1000$?

ORDER-DEPENDENT SUMS (CONTD.)

How do we solve for $N = 1000$?

Let D_N be the solution (e.g. $D_5 = 6$).

ORDER-DEPENDENT SUMS (CONTD.)

How do we solve for $N = 1000$?

Let D_N be the solution (e.g. $D_5 = 6$).

Define a recursion. Observe that

- any sum ends with a 1, 3 or 4.
- if the last term is i , the remaining sum to $N - i$.
- $D_N = D_{N-1} + D_{N-3} + D_{N-4}$

ORDER-DEPENDENT SUMS (CONTD.)

How do we solve for $N = 1000$?

Let D_N be the solution (e.g. $D_5 = 6$).

Define a recursion. Observe that

- any sum ends with a 1, 3 or 4.
- if the last term is i , the remaining sum to $N - i$.
- $D_N = D_{N-1} + D_{N-3} + D_{N-4}$

Also, $D_0 = D_1 = D_2 = 1, D_3 = 2$

ORDER-DEPENDENT SUMS (CONTD.)

How do we solve for $N = 1000$?

Let D_N be the solution (e.g. $D_5 = 6$).

Define a recursion. Observe that

- any sum ends with a 1, 3 or 4.
- if the last term is i , the remaining sum to $N - i$.
- $D_N = D_{N-1} + D_{N-3} + D_{N-4}$

Also, $D_0 = D_1 = D_2 = 1, D_3 = 2$

```
ordered_sum <- function(N) {  
  D <- rep(1,N); D[c(3,4)] <- c(2, 3)  
  for(i in 5:N) {  
    D[i] <- D[i-1] + D[i-3] + D[i-4] }  
  return(D[N]) }  
}
```

THE KNAPSACK PROBLEM

Given:

- a bag with (integer) capacity W lbs
- n types of objects, with integer weights (w_1, \dots, w_n) lbs and positive value (v_1, \dots, v_n)
- Unlimited objects of each type

Goal: Fill bag to maximize value $\mathbb{V}(W)$

THE KNAPSACK PROBLEM

Given:

- a bag with (integer) capacity W lbs
- n types of objects, with integer weights (w_1, \dots, w_n) lbs and positive value (v_1, \dots, v_n)
- Unlimited objects of each type

Goal: Fill bag to maximize value $\mathbb{V}(W)$

What is $\mathbb{V}(0)$?

How about $\mathbb{V}(1)$ and $\mathbb{V}(2)$?

THE KNAPSACK PROBLEM

Given:

- a bag with (integer) capacity W lbs
- n types of objects, with integer weights (w_1, \dots, w_n) lbs and positive value (v_1, \dots, v_n)
- Unlimited objects of each type

Goal: Fill bag to maximize value $\mathbb{V}(W)$

What is $\mathbb{V}(0)$?

How about $\mathbb{V}(1)$ and $\mathbb{V}(2)$?

Can we express $\mathbb{V}(i)$ in terms of $\mathbb{V}(j), j < i$?

THE KNAPSACK PROBLEM

Given:

- a bag with (integer) capacity W lbs
- n types of objects, with integer weights (w_1, \dots, w_n) lbs and positive value (v_1, \dots, v_n)
- Unlimited objects of each type

Goal: Fill bag to maximize value $\mathbb{V}(W)$

What is $\mathbb{V}(0)$?

How about $\mathbb{V}(1)$ and $\mathbb{V}(2)$?

Can we express $\mathbb{V}(i)$ in terms of $\mathbb{V}(j), j < i$?

$$\mathbb{V}(i) = \max_{j:w_j \leq i} \mathbb{V}(i - w_j) + v_j$$

A 2-DIMENSIONAL DYNAMIC PROGRAM

A DNA molecule is a sequence of nucleotides (A,T,G and C).

Want to align two DNA sequences

Similarity can suggest functionality of a newly sequenced gene

Russell Doolittle and colleagues found similarities between cancer-causing gene and normal growth factor (PDGF) gene

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

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

What is the best alignment?

A 2-DIMENSIONAL DYNAMIC PROGRAM

A DNA molecule is a sequence of nucleotides (A,T,G and C).

Want to align two DNA sequences

Similarity can suggest functionality of a newly sequenced gene

Russell Doolittle and colleagues found similarities between cancer-causing gene and normal growth factor (PDGF) gene

Simple sources of misalignment:

Substitution: A-A-C-T-G-G-A

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

Insertion: A-A-C-G-G-A

A-A-C-* -G-A

Deletion: A-A-C-T-* -G-A

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

SEQUENCE ALIGNMENT

Given two sequences:

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

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

What is the best alignment?

SEQUENCE ALIGNMENT

Given two sequences:

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

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

Define a distance between two sequences:

- Each substitution has a cost C_S
- Each insertion/deletion has a cost C_G (gap penalty)
- In practice, these can depend on the nucleotides

A-A-* -C-T-A-T-G-G-C-C-A

A-C-A-C-T-A-T-G-G-* -C-T

This alignment has cost $2C_S + 2C_G$.

DYNAMIC PROGRAMMING RECURSION

Consider aligning two strings S_1 and S_2 of length i and j :

$S_1 = \dots\text{-G-C-C-A}$ and $S_2 = \dots\text{-G-G-C-T}$

DYNAMIC PROGRAMMING RECURSION

Consider aligning to two strings S_1 and S_2 of length i and j :

$S_1 = \dots\text{-G-C-C-A}$ and $S_2 = \dots\text{-G-G-C-T}$

Three possibilities:

- The last two characters are matched:

...-A	$C_M(i, j) = \text{Cost}(i-1, j-1) +$ Cost of matching elements $S_1(i)$ and $S_2(j)$.
...-T	

- A gap in the first string:

...-*	$C_I(i, j) = \text{Cost}(i, j-1) +$ Cost of inserting gap after $S_2(j)$.
...-T	

- A gap in the second string:

...-A	$C_D(i, j) = \text{Cost}(i-1, j) +$ Cost of inserting gap after $S_1(i)$.
...-*	

The actual (best) cost:

$$\text{Cost}(i, j) = \min(C_M(i, j), C_I(i, j), C_D(i, j))$$

[<http://baba.sourceforge.net>]

BACKTRACKING

Forward recursion only returns cost of the best alignment.
What is this alignment?

Compute via a **backward trace**

BACKTRACKING

Forward recursion only returns cost of the best alignment.
What is this alignment?

Compute via a **backward trace**

Recall: $Cost(i, j) = \min (C_M(i, j), C_I(i, j), C_D(i, j))$

BACKTRACKING

Forward recursion only returns cost of the best alignment.
What is this alignment?

Compute via a **backward trace**

Recall: $Cost(i, j) = \min (C_M(i, j), C_I(i, j), C_D(i, j))$

- If $Cost(i, j) = C_M(i, j)$ then add $S_1(i)$ and $S_2(j)$ to the heads of strings 1 and 2 respectively, and decrement i and j .

BACKTRACKING

Forward recursion only returns cost of the best alignment.
What is this alignment?

Compute via a **backward trace**

Recall: $Cost(i, j) = \min (C_M(i, j), C_I(i, j), C_D(i, j))$

- If $Cost(i, j) = C_M(i, j)$ then add $S_1(i)$ and $S_2(j)$ to the heads of strings 1 and 2 respectively, and decrement i and j .
- If $Cost(i, j) = C_I(i, j)$ then add $S_1(i)$ to the head of strings 1, and decrement i .

Forward recursion only returns cost of the best alignment.
What is this alignment?

Compute via a **backward trace**

Recall: $Cost(i, j) = \min (C_M(i, j), C_I(i, j), C_D(i, j))$

- If $Cost(i, j) = C_M(i, j)$ then add $S_1(i)$ and $S_2(j)$ to the heads of strings 1 and 2 respectively, and decrement i and j .
- If $Cost(i, j) = C_I(i, j)$ then add $S_1(i)$ to the head of strings 1, and decrement i .
- If $Cost(i, j) = C_D(i, j)$ then add $S_2(j)$ to the head of strings 2, and decrement j .

[<http://baba.sourceforge.net>]

Overall algorithm: Needleman-Wunsch algorithm.

Cost (for sequences of length N and M):

- Forward pass: $O(NM)$ time (computations)
 $O(NM)$ space (memory)
- Backward pass: $O(N + M)$ time

We looked at dynamic programming to solve complicated looking problems by recursively solving simpler subproblems.

Next class we'll focus on a special problem, viz. Kalman filtering.