ECS 122A Lecture 7 Jasper Lee

Answering range queries

A nice trick from doing DP

Setting: An array $A[1..n]$

Prepare a "data structure" to answer

range queries $(i,j) \mapsto \sum_{k=i}^{j} A[k]$

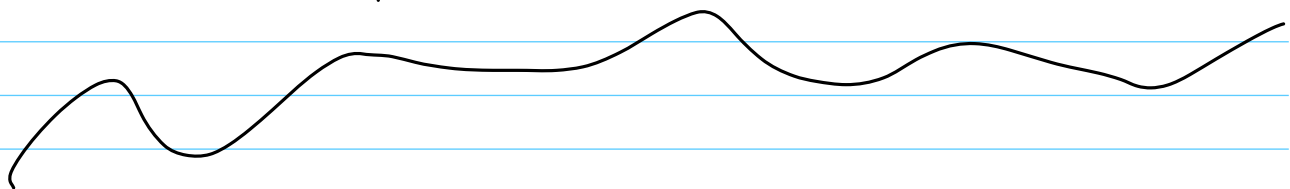Naive: Every query spend $O(j-i)$ time.

Better:

Observe $\sum_{k=i}^{j} A[k] = \sum_{k=1}^{j} A[k] - \sum_{k=1}^{i-1} A[k]$

So, compute prefix sum in $O(n)$ time

$$T[i] \doteq \sum_{k=1}^{i} A[k]$$

$$T[i+1] = T[i] + A[i+1]$$

Very simple, useful trick to have.

Longest Common Subsequence

Given a seq $S[1..n]$, (just an array)

a subsequence formed by removing some symbols.

e.g. $S = a\ b\ c\ d\ e\ f\ g$

subsequences: $a\ d\ f$
$a\ d\ f\ g$
$b\ c\ e$

So on.

2^n many subsequences.

LCS:

Given seq $A[1..m]$, $B[1..n]$

Find the longest subseq appearing in both A and B.

Exhaustive search takes $\exp(m)\ \exp(n)$ time...

Want: $O(mn)$ time — Fine-grained complexity conjectures there's no

LCS is "SETH-hard" ← $O(m^{.99} n^{.99})$ alg.

Given runtime, can afford mn size table.

Let's try the approach from before, use same problem on smaller input for table memory.

$T[i, j] \doteq$ LCS for $A[1..i]$
$\quad\quad\quad\quad\quad\quad\quad\quad\quad B[1..j]$

Obs: MUST
LCS either uses $\quad\quad\quad\quad A[i], B[j]$
$\lambda \quad$ or doesn't.
$\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad$ well need to
$\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad$ check if
$\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad A[i] = B[j].$

either doesn't use
$\quad A[i]$
or $\quad B[j]$

$T[i+1, j+1] = \max(\text{ if } A[i]=B[j] \text{ then } 1+T[i,j]$
$\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \text{else } 0,$
$\quad\quad\quad\quad\quad\quad\quad\quad T[i, j+1],$
$\quad\quad\quad\quad\quad\quad\quad\quad T[i+1, j])$

Fill in w/ 2d loop, like in # $\downarrow$, $\Rightarrow$ path
$\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad$ ``

Exercise: Do longest common substring
$\quad\quad\quad\quad$ in $O(mn)$ time

0-1 knapsack

Setting: knapsack has capacity C (integer)
$\quad\quad$ A list of n items each w/
$\quad\quad\quad\quad$ — Reward r_i
$\quad\quad\quad\quad$ — Weight w_i $\quad$ integer

Find a subset of items with total weight
$\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \leq C$
and maximize total reward.

"0-1" because either we take an item or not.
No multiple copies of the same item.

One of different formulations (others include integer reward instead of integer weight/capacity) require different algs!

Haven't talked about greedy, but can we?

No!

E.g. greedy for r_i/w_i fails

Item 1, $r_1 = 1$, $w_1 = 10^{-5}$
Item 2, $r_2 = 10$, $w_2 = 10$

Capacity $= 10$.

Can solve w/ DP.

First try

$T_0[i] = $ 0-1 knapsack on items $1..i$?

Can't possibly work, because of the capacity constraint.

New try, let's make the DP table 2-D!

$T[i, C] \doteq$ 0-1 knapsack on items $1..i$
under capacity C

Recurrence observation:

Opt soln <u>must</u> either

Use item $i \sim r_i + T[i-1, C-w_i]$
or
Not use item $i \sim T[i-1, C]$

$$T[i, C] = \max(r_i + T[i-1, C-w_i], \\ T[i-1, C])$$

How to fill table?
for $i = 1$ to n
     $c = 0$ to C $O(Cn)$ time

. . .

How much space? $O(Cn)$ space too

Space optimization
Only need to keep the row $T[i-1, \cdot]$
to compute $T[i, \cdot]$. $O(C)$ space.
Wrap up: What's the soln of original problem?
$T[n, C]$, by defn.