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

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

Better:

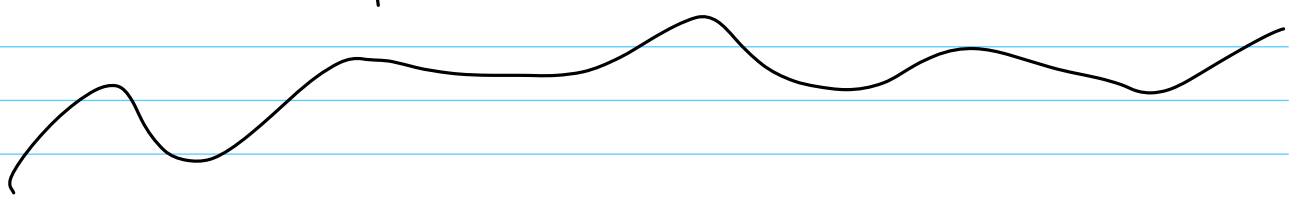
Observe $\sum_{k=i}^{j} A[k] = \sum_{k=1}^{j} A[k] - \sum_{k=1}^{i-1} A[k]$

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

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

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

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 $O(m^{.99}n^{.99})$ alg.

LCS is "SETH-hard" ←

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 $A[1..i]$, $B[1..j]$

Obs: MUST
LCS either uses $A[i], B[j]$ or doesn't.

well need to check if $A[i] = B[j]$.

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

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

Fill in w/ 2d loops like in #4, $\Rightarrow$ path "

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

0-1 Knapsack

Setting: Knapsack has capacity C (integer)

A list of n items each w/
- Reward r_i
- Weight w_i integer

Find a subset of items with total weight $\leq C$ 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!

keep track of info we need in a table / subproblem index

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

Recurrence observation:

Opt soln must either

Use item i ~ $r_i + T[i-1, C-w_i]$

or

Not use item i ~ $T[i-1, C]$

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

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.