CS3000

5/22 - Mon.

Admin

- Long HW2 due tom. 9pm
- Rec tom., no problem set → come with questions!
- next HW - Long 3, out thurs, due $\boxed{5/31}$
- Cracking the coding interview → OH

Agenda

1. Scheduling problem
2. DP solution to scheduling
3. Greedy solution to scheduling

Recap

DP mostly for optimization problems

Optimal substructure? Solution to a subproblem is contained in solution to bigger problem

DP solution

recursive formula

ex

$C[i,j] = C[i-1,j] + 1$

iterative

fill in C table bottom-up

1. Scheduling Problem

Activities that require use of a shared resource.
Only one activity can use the resource at a time.

$\boxed{\text{ex}}$

- access to a classroom
- a soccer field
- OS resources (ex: CPU)

Scheduling details

$S = \{a_1, a_2, \dots, a_n\}$ set of activities that want to use the resource

Every a_i has:

- s_i (start time)
- f_i (finish time)
- another activity can start at ~~at~~ or after a_i finishes

Activities a_i, a_j are compatible if $s_i \geq f_j$ or $s_j \geq f_i$

What ~~are~~ are we trying to optimize for?

- least amount of unused time
- ☆ most amount of activities ☆
- least amount of time to run the activities

Define set S_{ij} = set of compatible activities that start after a_i finishes and finish before a_j starts

Goal: maximize $|S_{ij}|$ → optimization problem!

ex

i	1	2	3	4	5	6
s_i	2	1	3	3	6	12
f_i	3	6	6	8	10	14

sorted by finish time!

Goal: maximize $|S_{ij}|$ (optimal S_{ij})

ex $S_{1,5} = \{a_3, a_4, a_5, a_6\}$ → start after a_1

$\cap$

$\{a_1, a_2, a_3\}$ → finish before a_5

$= \{a_3\}$

i	1	2	3	4	5	6
s_i	2	1	3	3	6	12
f_i	3	6	6	8	10	14

What is $S_{1,6}$? $\{a_3, a_5\}$ $|S_{1,6}| = 2$

now... remove a_3 $\leadsto$ $\{a_5\}$

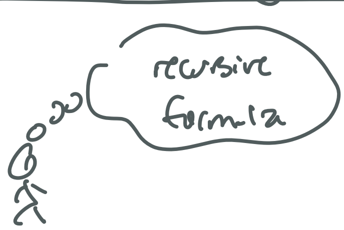
$S_{1,3} = \{\}$ $\rightarrow$!!

$S_{3,6} = \{a_5\}$

$$S_{1,6} = S_{1,3} + S_{3,6} + a_3$$

optimal substructure

2. DP scheduling solution



- Build a c table
- $c[i,j] = |S_{ij}|$ for the best/optimal S_{ij}

Recursive formula

$$C[i,j] = \begin{cases} 0 & \text{if } |S_{ij}| = 0 \\ \max \text{ of } a_k \in S_{ij} : C[i,k] + C[k,j] + 1 \end{cases}$$

i	1	2	3	4	5	6
s_i	2	1	3	3	6	12
f_i	3	6	6	8	10	14

ex: C table for ↑

	1	2	3	4	5	6
1	0	0	0	0	1 (3)	2 (3,5)
2		0	0	0	0	1 (5)
3			0	0	0	1 (5)
4				0	0	0
5					0	0
6						0

$\boxed{\text{Run-time: } \Theta(n^3) \quad \text{Space: } \Theta(n^2)}$

∴ better than brute force!

"can we do better?"

10:47

3. Greedy Solution to Scheduling

naive → divide+ conquer → DP → Greedy

Algorithmic techniques

Greedy

- always make the current best choice
- don't solve subproblems
- don't think recursively
- simple to implement! :)
- doesn't always work :(
- Is it efficient? TBD
- Tends to be used for optimization problems

Today's problem: Scheduling

- optimal (max) # of activities
- greedy: at any one time, pick the best compatible activity

- Always have a set S = set of compatible activities
- at any step, pick the best compatible activity and add to the set

What is the best activity to choose?

- Best = leaves most resources available
  = earliest finish time
  (more than one way though!)

Activity Algorithm:

- input: two arrays, s and f
  - s ↳ start times
  - f ↳ finish times

(correspond with each other!
a_i has s[i] start time
f[i] finish time)

- returns: set of activities (hopefully optimal)
  ↳ by indexes

ACTIVITY (s, f)

// sort s and f by finish time → pre process

$S = \{1\}$ → a_1 has earliest finish time

$k = 1$ → index of activity we just added

for $m = 2$ to f.length → iterate over activities

  if $s[m] \geq f[k]$ → Find compatible activity

    $S = S \cup \{m\}$ → update set S

    $k = m$ and index k

return S → return set of activities choices

i	1	2	3	4	5	6
s_i	2	1	3	3	6	12
f_i	3	6	6	8	10	14

ex

→ $\boxed{\text{already sorted!}}$

↳ $\boxed{\text{hopefully the answer!}}$

Steps of algo:

$S = \{1\}$

$k = 1$

$m = 2$

  is 2 compatible? no!

$m = 3$

is 3 compatible? yes!

$S = \{1, 3\}$

$m = 4$

is 4 compatible? no!

$m = 5$

is 5 compatible? yes!

$S = \{1, 3, 5\}$

$m = 6$

is 6 compatible? yes!

$S = \{1, 3, 5, 6\}$ → returned

Did we get an optimal solution? Yes :)

Run time? sort + loop

$\Theta(n \lg n) + \Theta(n)$ → Bound by Sort!

$\boxed{\Theta(n \lg n)}$

or runtime is $\boxed{\Theta(n)}$, assuming data is already sorted

Does this really work, or was our input lucky?

- greedy doesn't always work
- But in this case it does, where we sort by earliest finish time

How do we know it works?

S_k is any subproblem ($k \le n$)

S_k has a_m as earliest finish time

Let A_k be optimal solution to S_k

- Is $a_m \in A_k$? Done!
- Is $a_m \notin A_k$?

Let $a_j \in A_k$ with earliest finish time

Can we swap out a_j and replace with a_m?

Yes!

$f[m] \le f[j]$

a_m ~~does~~ can't conflict with an activity if a_j didn't conflict.

So, we have optimal solution $A_k = A_k - \{a_j\} \cup \{a_m\}$

- activity with earliest finish time is always in optimal solution