CSE 421 Section 5

Dynamic Programming

Administrivia

Announcements & Reminders

- HW3
  - If you think something was graded incorrectly, submit a regrade request!
- HW4
  - Due yesterday, 10/26
- HW5
  - Due Wednesday 8/1 @ 11:59pm
- Midterm Exam: **Wednesday November 8 in CSE2 G20 @ 6-7:30 pm**
  - Make sure you have it saved on your calendar!
  - If you can't make it, let us know and we will schedule a conflict exam!

Writing a Dynamic Programming Algo

Dynamic Programming

- Take recursive ideas from divide and conquer, but speed up finding the solution by optimizing the work by reordering and saving the results so we don't have to repeat anything!

- **Key idea**:
  - use English words to explain the output of the recursive function
  - write a recurrence for the output of the recursive function

- **Memoization**: save results of intermediate calculations so we don't need to repeat

Dynamic Programming

- **Recursion:** Use a recursive solution, but speed up finding the solution by optimizing the work by reordering and saving the results so we don't have to repeat anything!

- **Key idea**:
  - use English words to explain the output of the recursive function
  - write a recurrence for the output of the recursive function
- **Parameters:** Figure out all the parameters of the subproblems so we can store their answers in a nice structure without repeating

- **Order:** Order the subproblems so they are already finished when we need them.

The general strategy we've been using...

1. Read and Understand the Problem
2. Generate Examples
3. Produce a Baseline
4. Brainstorm and Analyze Possible Algorithms
5. Write an Algorithm
6. Show Your Algorithm is Correct
7. Optimize and Analyze the Run Time

The DP Strategy for Steps 4-7

4. Suppose brainstorming has led you to try Dynamic Programming...
5. Dynamic Programming:
   i. **Recurse:** Design a recursive solution for the problem
      - How are the subproblems for that solution defined?
      6. Correctness: Show that your recursive solution is correct.
   ii. **Parameters:** Determine the possible values of parameters in those subproblems and how to store them
   iii. **Order:** Design an iterative solution that computes them in the right order.
7. Optimize and Analyze the Run Time

Problem 1 – Lots of fun, with a normal sleep schedule

You are planning your social calendar for the month. For each day, you can choose to go to a social event or stay in and catch-up on sleep. If you go to a social event, you will enjoy yourself. But you can only go out for two consecutive days – if you go to a social event three days in a row, you'll fall too far behind on sleep and miss class.

Luckily, you have an excellent social sense, so you know exactly how much you will enjoy any of the social events, and have assigned each day an (integer) numerical happiness score (and you know you get 0 enjoyment from staying in and catching up on sleep). You have an array $H[]$ which gives the happiness you would get by going out each day. Your goal is to maximize the sum of the happinesses for the days you do go out, while not going out for more than two consecutive days.

1. Read and Understand the Problem

Problem 1.1 – Fun & Sleep

- Are there any **technical terms**, or words that seem technical?

- What is the **input type**? (Array? Graph? Integer? Something else?)

- What is your **return type**? (Integer? List?)

Spend a couple of minutes to figure these out

Problem 1.1 – Fun & Sleep

- Are there any **technical terms**, or words that seem technical?

- What is the **input type**? (Array? Graph? Integer? Something else?)

- What is your **return type**? (Integer? List?)

Problem 1.1 – Fun & Sleep

- Are there any **technical terms**, or words that seem technical?

  "consecutive" means in a row
  "maximize the sum of the happinesses" is semi-technical?

- What is the **input type**? (Array? Graph? Integer? Something else?)

- What is your **return type**? (Integer? List?)

Problem 1.1 – Fun & Sleep

- Are there any **technical terms**, or words that seem technical?

  "consecutive" means in a row
  "maximize the sum of the happinesses" is semi-technical?

- What is the **input type**? (Array? Graph? Integer? Something else?)

  `int[]`

- What is your **return type**? (Integer? List?)

Problem 1.1 – Fun & Sleep

- Are there any **technical terms**, or words that seem technical?

  "consecutive" means in a row
  "maximize the sum of the happinesses" is semi-technical?

- What is the **input type**? (Array? Graph? Integer? Something else?)

  `int[]`

- What is your **return type**? (Integer? List?)

  `int` (the maximum sum of happinesses)

2. Generate Examples

Good Examples Help!

- You should generate two or three sample instances and the correct associated outputs.

- It's a good idea to have some "abnormal" examples – consecutive negative numbers, very large negative numbers, only positive numbers, etc.

- *Note*: You should not think of these examples as debugging examples – null or the empty list is not a good example for this step. You can worry about edge cases at the end, once you have the main algorithm idea. You should be focused on the "typical" (not edge) case.

Problem 1.2 – Fun & Sleep

Generate two examples with their associated outputs. Put some effort into these! The more different from each other they are, the more likely you are to catch mistakes later.

Work through generating some examples, and then we'll go over it together!

Problem 1.2 – Fun & Sleep

Generate two examples with their associated outputs. Put some effort into these! The more different from each other they are, the more likely you are to catch mistakes later.

Problem 1.2 – Fun & Sleep

Generate two examples with their associated outputs. Put some effort into these! The more different from each other they are, the more likely you are to catch mistakes later.

[**2**, **2**, 1, **2**, **2**, 1, **2**, **2**] has a maximum happiness sum of 6

[**10**, 8, **15**, **9**, 3,11, **12**, **13**] has a maximum happiness sum of 59

3. Write the Dynamic Program

Problem 1.3 – Fun & Sleep

a) **Formulate the problem recursively** – what are you looking for (in English!!), and what parameters will you need as you're doing the calculation?

b) **Write a recurrence for solving the problem** you defined in the last part (the recurrence is for the answer, not the running time).

c) **What is your final answer** (e.g. what parameters for the recurrence do you need? Is it a single value or the max/min of a set of values?)?

d) **Give a brief justification for why your recurrence is correct**. You do not need a formal inductive proof, but your intuition will likely resemble one.

Start brainstorming some answers to these questions.

Problem 1.3 – Fun & Sleep

a) **Formulate the problem recursively** – what are you looking for (in English!!), and what parameters will you need as you're doing the calculation?

b) **Write a recurrence for solving the problem** you defined in the last part (the recurrence is for the answer, not the running time).

First, let's take some time to brainstorm about what the recurrence could be. What is our OPT finding? How many parameters do we need to calculate it? What are those parameters for?

Problem 1.3 – Fun & Sleep

a) **Formulate the problem recursively** – what are you looking for (in English!!), and what parameters will you need as you're doing the calculation?

Problem 1.3 – Fun & Sleep

a) **Formulate the problem recursively** – what are you looking for (in English!!), and what parameters will you need as you're doing the calculation?

As usual we consider how to deal with the last day assuming that we have solutions to problems involving the previous days. To know whether we can include the last day we need to know how many consecutive fun days we have taken immediately prior (0, 1, or 2).

Let OPT(j, k) be the most points we can earn in the array from $1..j$ (inclusive) where we have taken k consecutive days at the right end of the subproblem (e.g. if $k = 2$ then we have included elements j and $j - 1$ but not element $j - 2$). Per the problem, we only allow $k \in \{0,1,2\}$ and $j \in \{0,1, \dots, n\}$.

Problem 1.3 – Fun & Sleep

b) **Write a recurrence for solving the problem** you defined in the last part (the recurrence is for the answer, not the running time).

Problem 1.3 – Fun & Sleep

b) **Write a recurrence for solving the problem** you defined in the last part (the recurrence is for the answer, not the running time).

$$\mathrm{OPT}(j,0) = \begin{cases} 0 & \text{if } j = 0 \\ \max\big(OPT(j-1,0), OPT(j-1,1), OPT(j-1,2)\big) & \text{if } j \geq 1 \end{cases}$$

$$OPT(j,1) = \begin{cases} -\infty & \text{if } j = 0 \\ A[j] + OPT(j-1,0) & \text{if } j \geq 1 \end{cases}$$

$$OPT(j,2) = \begin{cases} -\infty & \text{if } j \leq 1 \\ A[j] + OPT(j-1,1) & \text{if } j \geq 2 \end{cases}$$

Alternative:

$$OPT(j,2) = \begin{cases} -\infty & \text{if } j \leq 1 \\ A[j] + A[j-1] + OPT(j-2,0) & \text{if } j \geq 2 \end{cases}$$

Problem 1.3 – Fun & Sleep

c) **What is your final answer** (e.g. what parameters for the recurrence do you need? Is it a single value or the max/min of a set of values?)?

Problem 1.3 – Fun & Sleep

c) **What is your final answer** (e.g. what parameters for the recurrence do you need? Is it a single value or the max/min of a set of values?)?

$$\max(OPT(n, 0), OPT(n, 1), OPT(n, 2))$$

Problem 1.3 – Fun & Sleep

d) **Give a brief justification for why your recurrence is correct**. You do not need a formal inductive proof, but your intuition will likely resemble one.

Problem 1.3 – Fun & Sleep

Give a brief justification for why your recurrence is correct. You do not need a formal inductive proof, but your intuition will likely resemble one.

For $OPT(j,0)$: If $j = 0$ there are no elements to include and the empty solution with value 0 is the best one can do. Otherwise, $j \geq 1$ and we need to skip element j. Any one of the options for the first j-1 elements is possible, so we should select the best of the 3 options which are given by $OPT(j-1,0)$, $OPT(j-1,1)$ and $OPT(j-1,2)$.

For $OPT(j,1)$, we must include both $A[j]$ but not $A[j-1]$. This is impossible for $j = 0$ since there is no such element so we encode this impossibility with value $-\infty$ so that it can never be part of an optimal solution. Otherwise, we need to add $A[j]$ to the most happiness from days $1, \ldots, j-1$ where we do not include $A[j-1]$, which is the definition of $OPT(j-1,0)$.

For $OPT(j,2)$, we must include both $A[j]$ and $A[j-1]$ but not $A[j-2]$. This is impossible if $j = 0$ or $j = 1$ since there are no such elements so we encode this impossibility with value $-\infty$ so that it can never be part of an optimum solution. This solution must includes $A[j]$ plus the solution giving the most happiness while including $A[j-1]$ but not $A[j-2]$ which is precisely given by the definition of $OPT(j-1,1)$.

Problem 1.3 – Fun & Sleep (An extra aside)

b) **Write a recurrence for solving the problem** you defined in the last part (the recurrence is for the answer, not the running time).

$$\mathrm{OPT}(j,0) = \begin{cases} 0 & \text{if } j = 0 \\ \max\big(OPT(j-1,0), OPT(j-1,1), OPT(j-1,2)\big) & \text{if } j \geq 1 \end{cases}$$

$$OPT(j,1) = \begin{cases} -\infty & \text{if } j = 0 \\ A[j] + OPT(j-1,0) & \text{if } j \geq 1 \end{cases}$$

$$OPT(j,2) = \begin{cases} -\infty & \text{if } j \leq 1 \\ A[j] + OPT(j-1,1) & \text{if } j \geq 2 \end{cases}$$

Can you combine the last two lines in a general form that would be convenient if you could have anything up to f fun days in a row?

Problem 1.3 – Fun & Sleep (An extra aside)

b) **Write a recurrence for solving the problem** you defined in the last part (the recurrence is for the answer, not the running time).

$$\mathrm{OPT}(j,0) = \begin{cases} 0 & \text{if } j = 0 \\ \max\big(OPT(j-1,0), OPT(j-1,1), OPT(j-1,2)\big) & \text{if } j \geq 1 \end{cases}$$

$$OPT(j,1) = \begin{cases} -\infty & \text{if } j = 0 \\ A[j] + OPT(j-1,0) & \text{if } j \geq 1 \end{cases}$$

$$OPT(j,2) = \begin{cases} -\infty & \text{if } j \leq 1 \\ A[j] + OPT(j-1,1) & \text{if } j \geq 2 \end{cases}$$

Combine the last two lines and write a general form as:

$$\text{For } k = 1, \ldots f\text{:}\quad OPT(j,k) = \begin{cases} -\infty & \text{if } j \leq k-1 \\ A[j] + OPT(j-1,k-1) & \text{if } j \geq k \end{cases}$$

4. Analyze the Dynamic Program

Problem 1.4 – Fun & Sleep

a) Describe the **parameters** for all the subproblems and how you will store them.

b) Describe the **order** for evaluating your subproblems.

c) Write the pseudocode for your iterative algorithm

d) State and justify the **running time** of an iterative solution.

Start brainstorming some answers to these questions.

Problem 1.4 – Fun & Sleep

a) Describe the **parameters** for all the subproblems and how you will store them.

b) Describe the **order** for evaluating your subproblems.

c) Write the pseudocode for your iterative algorithm

d) State and justify the **running time** of an iterative solution.

Problem 1.4 – Fun & Sleep

a) Describe the **parameters** for all the subproblems and how you will store them.

The parameters have $j = 0,1, \dots, n$ and $k = 0,1,2$.
We need an $(n+1) \times 3$ array, where entry j, k is $\mathrm{OPT}(j, k)$.

b) Describe the **order** for evaluating your subproblems.

c) Write the pseudocode for your iterative algorithm

d) State and justify the **running time** of an iterative solution.

Problem 1.4 – Fun & Sleep

a) Describe the **parameters** for all the subproblems and how you will store them.

The parameters have $j = 0,1, \ldots, n$ and $k = 0,1,2$.
We need an $(n+1) \times 3$ array, where entry j, k is $\mathrm{OPT}(j, k)$.

b) Describe the **order** for evaluating your subproblems.

Initialize case for j=0.
Outer loop j from 1 to n. Inside the loop evaluate each of the 3 cases.

Problem 1.4 – Fun & Sleep

c) Write the pseudocode for your iterative algorithm

```
OPT[0,0]= 0; OPT[0,1] = -∞; OPT[0,2]= -∞
for j = 1 to n do
    OPT[j,0] = max(OPT[j-1,0],OPT[j-1,1],OPT[j-1,2])
    OPT[j,1] = A[j]+OPT[j-1,0]
    if j==1 then
        OPT[j,2]= -∞
    else
        OPT[j,2] = A[j]+OPT[j-1,1]
return max(OPT[n,0],OPT[n,1],OPT[n,2])
```

Problem 1.4 – Fun & Sleep

d) State and justify the **running time** of an iterative solution.

In each of the n iterations, we check at most 3 entries, and we have 3 entries to fill, so our total running time is $\mathcal{O}(n)$.

That's All, Folks!

Thanks for coming to section this week!
Any questions?