CSE 421 Section 5

Dynamic Programming

Administrivia

Announcements & Reminders

- HW3
  - If you think something was graded incorrectly, submit a regrade request!
- HW4
  - Due yesterday, 2/1
- HW5
  - Due Wednesday 2/8 @ 11:59pm
- Midterm Exam: **Monday February 13 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

The Strategy (SLIGHTLY DIFFERENT FOR DP)

1. Read and Understand the Problem
2. Generate Examples
3. Write the Dynamic Program
4. Analyze the Dynamic Program

Dynamic Programming Process (from lecture)

This is what we'll do in parts 3 and 4 of our strategy:

1. Define the object you're looking for.
2. Write a recurrence to say how to find it.
3. Design a memoization structure.
4. Write an iterative algorithm.

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?)

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.

$[\mathbf{2}, \mathbf{2}, 1, \mathbf{2}, \mathbf{2}, 1, \mathbf{2}, \mathbf{2}]$ has a maximum happiness sum of 6

$[\mathbf{10}, 8, \mathbf{15}, \mathbf{9}, 3, 11, \mathbf{12}, \mathbf{13}]$ 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?

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

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).

$$\mathrm{OPT}(i,j) = \begin{cases} A[i] + \mathrm{OPT}(i-1,1) & \text{if } i > 1, j = 2 \\ A[i] + \max_y \mathrm{OPT}(i-2,y) & \text{if } i > 2, j = 1 \\ \max_y \mathrm{OPT}(i-1,y) & \text{if } i > 1, j = 0 \\ A[i] & \text{if } i = 1,2, j = 1 \\ 0 & \text{if } i = 1, j = 0 \\ -\infty & \text{otherwise} \end{cases}$$

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?)?

$$\max_j \mathrm{OPT}(n, j)$$

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 $i > 1, j = 2$, we must include both $A[i]$ and $A[i-1]$, but not $A[i-2]$, so we need to add $A[i]$ to the most points among $1, \dots, i-1$ where we include $A[i-1]$ but not $A[i-2]$, which is the definition of $\mathrm{OPT}(i-1, 1)$.

For $i > 2, j = 1$, we must include $A[i]$ but not $A[i-1]$. We therefore want to add $A[i]$ the maximum points we can earn from $1, \dots, i-2$. Since we skip element $i-1$, we have no requirement on whether to include $i-2$ or not, and just desire the maximum number of points among $1, \dots, i-2$; the best sequence either excludes $\mathrm{A}[i-2]$, includes $A[i-2]$ but not $A[i-3]$, or includes both $A[i-2], A[i-3]$ but not $A[i-4]$, thus we want the max of $\mathrm{OPT}(i-2, 0)$, $\mathrm{OPT}(i-2, 1)$, $OPT(i-2, 2)$ added to $A[i]$

If $j = 0$, we simply need to skip $A[i]$, and want the maximum number of points for $1, \dots, i-1$ with no restrictions. We thus check all three options for the end of the array (none, one, or two elements at the right).

For the base/edge cases: for $i = 1,2,\ j = 1$, our only choice is to take $A[i]$ and for $i = 1, j = 0$, we must not take any elements. All other combinations of i, j are invalid (there are no elements to take, or j is large enough we would have to take more elements than there are) so we choose $-\infty$ which will never enter into a max calculation.

4. Analyze the Dynamic Program

Problem 1.4 – Fun & Sleep

a) Describe a **memoization** structure for your algorithm.

b) Describe a **filling order** for your memoization structure.

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

Start brainstorming some answers to these questions.

Problem 1.4 – Fun & Sleep

a) Describe a **memoization** structure for your algorithm.

b) Describe a **filling order** for your memoization structure.

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

Problem 1.4 – Fun & Sleep

a) Describe a **memoization** structure for your algorithm.

We need an $n \times 3$ array, where entry i, j is $\mathrm{OPT}(i, j)$.

b) Describe a **filling order** for your memoization structure.

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

Problem 1.4 – Fun & Sleep

a) Describe a **memoization** structure for your algorithm.

We need an $n \times 3$ array, where entry i, j is $\mathrm{OPT}(i, j)$.

b) Describe a **filling order** for your memoization structure.

Outer loop i from 1 to n
Inner loop j from 0 to 2

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

Problem 1.4 – Fun & Sleep

a) Describe a **memoization** structure for your algorithm.

We need an $n \times 3$ array, where entry i, j is $\mathrm{OPT}(i, j)$.

b) Describe a **filling order** for your memoization structure.

Outer loop i from 1 to n
Inner loop j from 0 to 2

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

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

Problem 2:
Longest Increasing Subsequence AGAIN

Problem 2 – Longest Increasing Subsequence

We've already seen a recurrence for Longest Increasing Subsequence. Let's write another!

As before, [10, −2, 5, 0, 3, 11, 8] has a longest increasing subsequence of 4 elements: [−2, 0, 3, 8]

Problem 2.1 – Write the Dynamic Program

a) Formulate the problem recursively – what are you looking for (in English!!), and what parameters will you need as you're doing the calculation? To make sure you get a different solution than the one from class, you should ask yourself to answer the question "what's the longest increasing subsequence where the first included element is the one at index i, and how would I find that?"

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.

Problem 2.1 – Write the Dynamic Program

a) Formulate the problem recursively – what are you looking for (in English!!), and what parameters will you need as you're doing the calculation? To make sure you get a different solution than the one from class, you should ask yourself to answer the question "what's the longest increasing subsequence where the first included element is the one at index i, and how would I find that?"

Problem 2.1 – Write the Dynamic Program

a) Formulate the problem recursively – what are you looking for (in English!!), and what parameters will you need as you're doing the calculation? To make sure you get a different solution than the one from class, you should ask yourself to answer the question "what's the longest increasing subsequence where the first included element is the one at index i, and how would I find that?"

Let LISAlt(i) be the length of the longest increasing subsequence of $A[]$ where element i is the first element of the subsequence.

Problem 2.1 – Write the Dynamic Program

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 2.1 – Write the Dynamic Program

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

$$\mathrm{LISAlt}(i) = \begin{cases} 1 & \text{if } i = n \\ 1 + \max_{j>i} \mathbb{I}\left[A[i] < A[j]\right] \cdot \mathrm{LISAlt}(j) & \end{cases}$$

Problem 2.1 – Write the Dynamic Program

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 2.1 – Write the Dynamic Program

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?)?

$$\max_i \text{LISAlt}(i)$$

Problem 2.1 – Write the Dynamic Program

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 2.1 – Write the Dynamic Program

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.

For the base case, since n is the farthest right element, it is the only element in a subsequence starting from that location.

If we begin at element i, then either it is the only element or there is an element after. The recurrence checks all elements after – if they are the second element in that sequence, they must be after i, have the element be greater than $A[i]$. That new location j will then start the rest of the increasing subsequence, so making all those recursive calls suffices to find the best one

Problem 2.2 – Analyze the Dynamic Program

a) Describe a memoization structure for your algorithm.

b) Describe a filling order for your memoization structure.

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

Start brainstorming some answers to these questions.

Problem 2.2 – Analyze the Dynamic Program

a) Describe a memoization structure for your algorithm.

Problem 2.2 – Analyze the Dynamic Program

a) Describe a memoization structure for your algorithm.

We need a (1D) array of size n.

Problem 2.2 – Analyze the Dynamic Program

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 2.2 – Analyze the Dynamic Program

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

We fill from n down to 1.

Problem 2.2 – Analyze the Dynamic Program

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 2.2 – Analyze the Dynamic Program

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?)?

Creating entry i requires checking $i - 1$ recursive calls. Since we have n entries, we need $\mathcal{O}(n^2)$ time.

That's All, Folks!

Thanks for coming to section this week!
Any questions?