Analysis of Algorithms I Midterm Review Session 2/4/17

Presented by Raymond Xu

Dynamic Programming (DP)

- optimal substructure
- overlapping subproblems

My DP Strategy

1) Identify that the problem can be solved with a DP approach

- Not every problem is DP obviously
- It might not be immediately apparent which problems are DP on the midterm
- Familiarize yourself with common DP problem themes

2) Determine the dimensions of the DP matrix

- 1D, 2D, (3D?!, etc.)
- Each dimension should be related to a variable in the problem
- Have to think ahead to the next steps to figure this out, so these steps aren't exactly sequential
- Usually only a few options for dimensions

3) Define what each entry in the DP matrix means and initialize it

- Write out, for example, $\mathrm{OPT[i] = the\ minimum\ cost\ it\ takes\ to\ reach\ hotel\ i\ from\ the\ start}$
- This helps you ground your logic when you try and figure out the recurrence
- Then initialize entries to their appropriate values

4) Define how to compute the solution to the problem assuming you have filled out the DP matrix

- It might be just the value at an entry in your matrix, it might be the min/max over many entries
- Usually not too much work to reach the solution after the matrix is filled

5) Figure out the recurrence

- Probably the hardest part of solving DP problems
- Think about how the optimal solution at i can be produced using the previous optimal solutions
- Usually requires comparisons of options and picking the min/max
- Examples: look at just the previous OPT, look at all the previous OPTs, look at neighboring OPTs

6) Determine the time and space complexity

- Easy once you’ve gotten here
- Space complexity is usually just the size of the matrix
- Time complexity is also easy since you know that for each entry in the matrix you did some work to fill it in. So take the number of entries in the matrix and multiply that by the work done at each step

7) Reconstruct the solution

- If asked for, you should be able to modify your algorithm so that you can reconstruct the instance that yields the optimal value
- A common pattern is to store back-pointers at each step to the step that you came from. Then, follow the pointers back from the solution entry to build the instance (linked list)

8) Prove your algorithm is correct

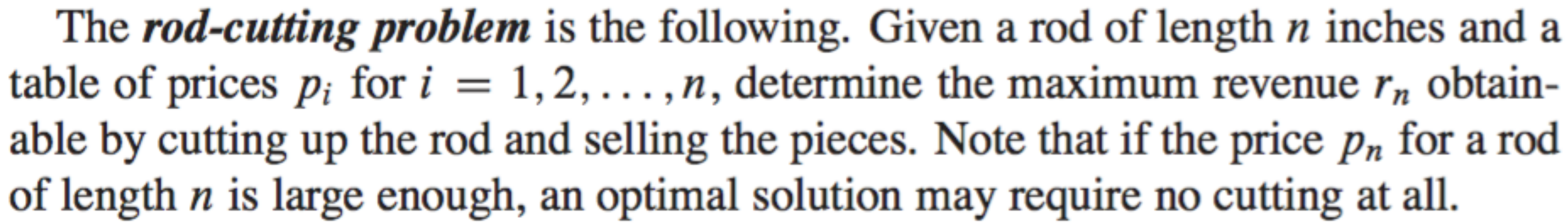
– If you've figured out the recurrence and have convinced yourself that it's correct, an inductive proof should be easy

1) Identify that the problem can be solved with a DP approach
2) Determine the dimensions of the DP matrix
3) Define what each entry in the DP matrix means and initialize it
4) Define how to compute the solution to the problem assuming you have filled out the DP matrix
5) Figure out the recurrence
6) Determine the time and space complexity
7) Reconstruct the solution
8) Prove your algorithm is correct

How do I get better at DP?

Practice Problems

Problem 1: Rod Cutting



The ***rod-cutting problem*** is the following. Given a rod of length n inches and a table of prices p_i for $i = 1, 2, \ldots, n$, determine the maximum revenue r_n obtainable by cutting up the rod and selling the pieces. Note that if the price p_n for a rod of length n is large enough, an optimal solution may require no cutting at all.

(CLRS p360)

Problem 1: Rod Cutting

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

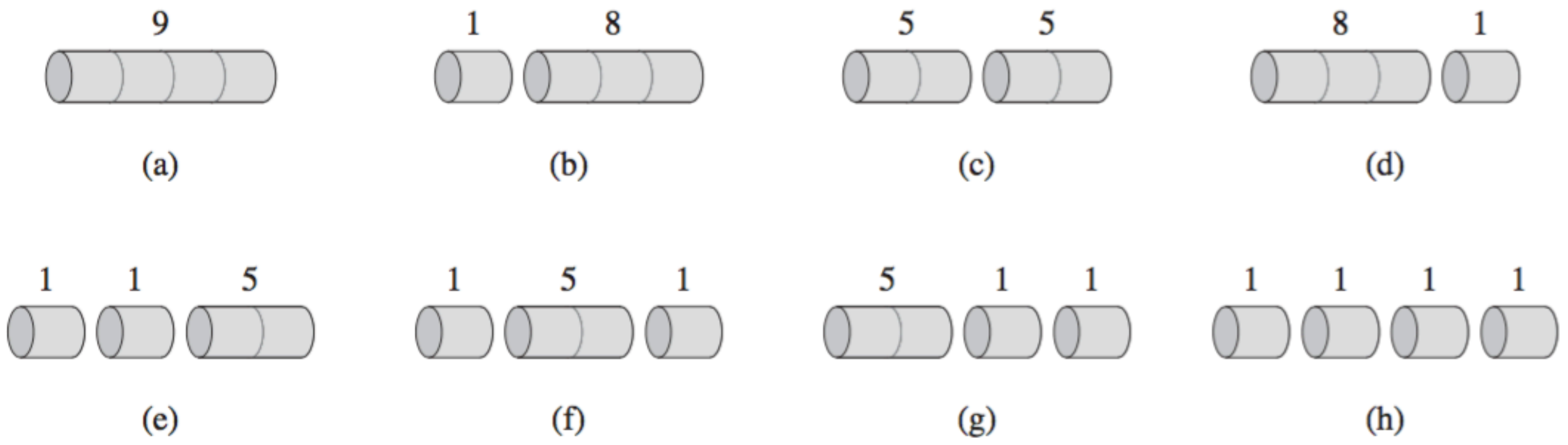


Figure 15.2 The 8 possible ways of cutting up a rod of length 4. Above each piece is the value of that piece, according to the sample price chart of Figure 15.1. The optimal strategy is part (c)—cutting the rod into two pieces of length 2—which has total value 10.

Problem 1: Rod Cutting

- Optimal substructure
  - after you’ve cut the rod, you can treat each of the two pieces as their own problems
  - the optimal revenue attainable for small lengths inform the optimal revenue attainable for larger lengths
- Basic recursive solution is $\mathrm{O}(2^{\mathrm{n}})$
  - repeats work

Problem 1: Rod Cutting

$$r_n = \max_{1 \le i \le n} (p_i + r_{n-i})$$

- How do you get the solution after you’ve filled out the array?
- How much work do you do at each step?
- Time and space complexity?

Problem 2: Coin Change

- Given an infinite supply of each of $\{c_1, c_2, \dots, c_m\}$ valued coins, what is the minimum number of coins needed to make change for exactly V cents?

Problem 2: Coin Change

- Given an infinite supply of each of $\{c_1, c_2, \ldots, c_m\}$ valued coins, what is the minimum number of coins needed to make change for exactly V cents?

- Greedy solution works for some cases (e.g. $\{25, 10, 5, 1\}$)
- DP solution required for generalized problem (e.g. $\{5, 4, 1\}$)

Problem 2: Coin Change

- Make change for every value <= n, smallest to largest

- $\mathrm{OPT}[0] = 0$
- $\mathrm{OPT}[n] = \min(\mathrm{OPT}[n - c] + 1)$
  - $\text{for all coins with value } c <= n$

Problem 2: Coin Change

- Time complexity?
- Space complexity?
- Reconstruction?

Problem 3: Unbounded Knapsack

- Given a knapsack with capacity W and a set of n items, each with a weight and value, what is the maximum value that can be stored in the knapsack?
- Assume all weights are nonnegative integers, and that you are allowed to take any number of each item (Unbounded Knapsack Problem)

Problem 3: Unbounded Knapsack

- Use a DP array of length W + 1

- OPT[0] = 0
- OPT[n] = max(OPT[n – w_i] + v_i) for all w_i <= n

- Solution is at OPT[W]
- O(nW) runtime

Problem 4: 0-1 Knapsack

- Given a knapsack with capacity W and a set of n items, each with a weight and value, what is the maximum value that can be stored in the knapsack?

- Now you can only take 0 or 1 of each item (0-1 Knapsack Problem)

Problem 4: 0-1 Knapsack

- Now we need a 2D DP matrix
  - (n + 1) by (W + 1)

- OPT[0, w] = 0
- OPT[i, w] = OPT[i – 1, w] if $w_i > w$
- OPT[i, w] = max(OPT[i – 1, w],
  OPT[i – 1, w – w_i] + v_i) if $w_i <= w$

Solution is at OPT[n, W]
O(nW) runtime

DP Practice

- Review the slides
- Read the textbook

- Do problems from the practice session
- Do problems in the textbook
- Do problems online

- Spend time thinking about the problems, don't just look up solutions

Analysis of Algorithms I Midterm Review Session

2/4/17

Presented by Raymond Xu