Programming Challenges

Week 6 - Dynamic Programming

Claus Aranha
caranha@cs.tsukuba.ac.jp

College of Information Siences

2015-05-29

Last updatet May 28, 2015

Dynamic Programming – Outline

- Extremely common on programming challenges;
- The basic idea is quite simple, but sometimes it is hard to grasp;
- Let’s use an example based approach;

First Example: Coins to a Sum

Sample Problem

We have many coins worth 1, 3 and 5 yenes. How can we combine these coins to add S using the minimum amount of coins?

... No, using modulo does not count

First Example: Coins to a Sum

S	1	2	3	4	5	6	7	8	9	10	11
C											

- How many coins are needed for $S = 1$? for $S = 2$? for $S = 3$? for $S = i$?

First Example: Coins to a Sum

S	1	2	3	4	5	6	7	8	9	10	11
C	1	2									

- How many coins are needed for $S = 1$? for $S = 2$? for $S = 3$? for $S = i$?

First Example: Coins to a Sum

S	1	2	3	4	5	6	7	8	9	10	11
C	1	2	1	2	1						

- How many coins are needed for $S = 1$? for $S = 2$? for $S = 3$? for $S = i$?

First Example: Coins to a Sum

S	1	2	3	4	5	6	7	8	9	10	11
C	1	2	1	2	1	2	3	2	3	2	3

- How many coins are needed for $S = 1$? for $S = 2$? for $S = 3$? for $S = i$?

First Example: Coins to a Sum

S	1	2	3	4	5	6	7	8	9	10	11
C	1	2	1	2	1	2	3	2	3	2	3

- How many coins are needed for $S = 1$? for $S = 2$? for $S = 3$? for $S = i$?
- If we know the values of $S_{i-1}, S_{i-3}, S_{i-5}$, we can use those to calculate S_i;

Basic Idea for Dynamic Programming

Table of Partial Results

We want to create a table with all partial results up to the desired value.

- How to define the table?
- How to fill the table's values?

More Coins to a Sum

- Filling the table forward:

S	0	1	2	3	4	5	6	7	8	9	10	11
C	-	1	2	1	2	1	2	3	2	3	2	3

- Filling the table backwards:
- Filling the table by coin:

More Coins to a Sum

- Filling the table forward:
- Filling the table backwards:

S	0	1	2	3	4	5	6	7	8	9	10	11
C	3	2	3	2	3	2	1	2	1	2	1	-

- Filling the table by coin:

More Coins to a Sum

- Filling the table forward:
- Filling the table backwards:
- Filling the table by coin:

S	0	1	2	3	4	5	6	7	8	9	10	11
C1	0	1	2	3	4	5	6	7	8	9	10	11

More Coins to a Sum

- Filling the table forward:
- Filling the table backwards:
- Filling the table by coin:

S	0	1	2	3	4	5	6	7	8	9	10	11
C1	0	1	2	3	4	5	6	7	8	9	10	11
C2	0	1	2	1	2	3	2	3	4	3	4	5

More Coins to a Sum

- Filling the table forward:
- Filling the table backwards:
- Filling the table by coin:

S	0	1	2	3	4	5	6	7	8	9	10	11
C1	0	1	2	3	4	5	6	7	8	9	10	11
C2	0	1	2	1	2	3	2	3	4	3	4	5
C3	0	1	2	1	2	1	2	3	2	3	2	3

States and Decisions

States

A state is a partial solution. Each state is independent: you don't need to know how the program arrived at that state, just its current value.

Decisions

Each state has a number of possible transitions from itself to other states. These are the actions that can be taken in the problem.

The Key for Dynamic programming is being able to define what are the states, and what are the decisions in a problem.

Example 2: Longest Subsequence

Problem Description

Given a sequence S of integers, find the largest subsequence Z where $z_0 < z_1 < z_2 < \cdots < z_n$

Values:

2 4 3 8 5 7 4 9

Example 2: Longest Subsequence

Problem Description

Given a sequence S of integers, find the largest subsequence Z where $z_0 < z_1 < z_2 < \cdots < z_n$

Values:

2	4	3	8	5	7	4	9

Length:

1	2	2

Last:

0	1	1

Example 2: Longest Subsequence

Problem Description

Given a sequence S of integers, find the largest subsequence Z where $z_0 < z_1 < z_2 < \cdots < z_n$

Values:

2	4	3	8	5	7	4	9

Length

1	2	2	3	3

Last:

0	1	1	3	3

Example 2: Longest Subsequence

Problem Description

Given a sequence S of integers, find the largest subsequence Z where $z_0 < z_1 < z_2 < \cdots < z_n$

Values:

2	4	3	8	5	7	4	9

Length

1	2	2	3	3	4	3	5

Last:

0	1	1	3	3	5	3	6

Example 2: Longest Subsequence

Algorithm

- State 0: Length = 1, Parent = 0;
- State i:
  - Length S_i = 1; Parent = 0;
  - For (j = 0 to i-1):
  - if ($S_j < S_i$) and Length(S_j) >= Length(S_i):
  - Length(S_i) = Length(S_j)+1
  - Parent(S_i) = S_j

Another kind of Recursion

Dynamic Programming problems use a very similar mindset as recursive problems: You want to find a recurrence function, and use it to calculate new solutions.

- It might be useful to store a “parent” and “decision” arrays, to restore the solution that was found.

When is Dynamic Programming Useful?

- A “quick” way to list all possible solution combinations;
- States are mostly independent;
- Combination of States/Transitions is polinomial;
- Table makes it easier to prune repetitions;

- Keep in mind the number of transitions!

Example 3: Apples

Problem Description

A farm has apples spread on a grid. You want to find the South/West path on the grid with the maximum number of apples.

Start: Top Left, End: Bottom Right;

2	3		3	5	5		1	
		1	2	2	3			2
2		3		2		2	2	
	3	2					3	4
	2		3		3			3
	2			3			3	

Example 3: Apples

Solving the Problem

- What are the states?
- What are the transitions?

Who can solve this first?

2	3		3	5	5		1	
		1	2	2	3			2
2		3		2		2	2	
	3	2					3	4
	2		3		3			3
	2			3			3	

Example 3.5: SUPER Apples

Problem Description

The Problem is the same as before, but now you have to cross the field three times:

1. Top Left to Bottom Right;
2. Bottom Right to Top Left;
3. Top Left to Bottom Right;

Give me your ideas!

Example 3.5: SUPER Apples

Let's try to solve it!

2	3		3	5	5		1	
		1	2	2	3			2
2		3		2		2	2	
	3	2					3	4
	2		3		3			3
	2			3			3	

Example 3.5: SUPER Apples

Hint 1

Hint 2

Hint 3

Example 3.5: SUPER Apples

Hint 1

Does direction make a difference?

Hint 2

Hint 3

Example 3.5: SUPER Apples

Hint 1

Does direction make a difference?

Hint 2

Does time make a difference?

Hint 3

Example 3.5: SUPER Apples

Hint 1

Does direction make a difference?

Hint 2

Does time make a difference?

Hint 3

Can/should the 3 pathes cross?

Example 3.5: SUPER Apples

Let's try to solve it!

2	3		3	5	5		1	
		1	2	2	3			2
2		3		2		2	2	
	3	2					3	4
	2		3		3			3
	2			3			3	

Another Idea about Dynamic Programming

If the problem sounds too complex, try to change the states around, or reduce it to a simpler problem.

Example 4: Money Djikstra

Problem Definition

We want to go from A_0 to A_n in an undirected graph. Every vertice has a cost C_i to enter that vertice.

We start with total money S_0, and we cannot enter a vertice if the cost of that vertice is bigger than our current money.

Find the shortest path in the graph. If more than one path have the same length, choose the one that spends the least money.

Example 5: Edit Distance

Problem Definition

Calculate the minimum cost to transform string *a* into string *b*, using the operations “INSERT”, “DELETE”, “CHANGE” and “NOOP”. The NOOP operation has cost 0, the other operations have cost 1.

This week's problems

- Direct Subsequences
- Adventures in Moving IV
- Chopsticks
- Unidirectional TSP