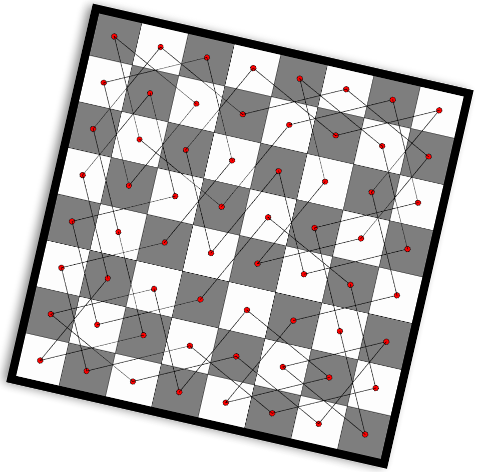
Dynamic Programming

INGI2266

Pierre Schaus



Interactive Questions

- www.wooclap.com/PDKMAJ

Typical Exam Questions

- Every year I ask DP question.
- This question is usually very badly solved by the students for various reasons.
- This session is to help you improve your skills and prepare better at the exam for this questions.
- The lecture will be very interactive with various exercises.
- The objective is mainly to practice, don’t be afraid to make mistakes.
- Rmq: DP questions are very frequent at programming interviews (Google, Amazon, Apple, etc).

Exam Question August 2016

Given (1) an arrangement $S=[s_1, . . . , s_n]$ of nonnegative numbers (2) an integer k, the objective is to partition S into k or fewer ranges, to minimize the maximum sum over all the ranges without reordering the numbers.

Example: S = [1, 2, 3, 4, 5, 6, 7, 8, 9] and k = 3.

An optimal partition into contiguous ranges is

[1, 2, 3, 4, 5], [6, 7], **[8, 9]**

with the largest one having a sum of **17**.

Just to be sure every-body follows

S = [1, 2, 6, 3, 1, 4, 5, 6, 7, 8, 5] and k = 4.

What is the optimal value?

1: 13

2: 9

3: 10

4: 15

5: 12

Subquestions

- Formulate this problem as a dynamic program. Write recurrence equations (don’t forget the base-cases)

- Sketch the code to solve it.

- What is the time complexity to solve this dynamic program (justify).

- Illustrate the execution and solution of your dynamic program on the following arrangement S = [10,2,3,4,5,1,7,8,4].

Subquestions

- Formulate this problem as a dynamic program.Write recurrence equations (don’t forget the base-cases)

Typical Mistakes:

A recurrence equation is given but it is not understandable. The range of parameters is not explained and what they represent is not explained.

Example of bad (incomplete) answer

$$O(i,l) = \max_j \min\left((s_i + s_{i+1} \ldots + s_j), O(i, l-1)\right)$$

What is i,l ?

What is the range of j

What does O represent?

No base case, what are the ranges for i,l? Can it be negative?

- Unfortunately you should get a grade of zero for this answer.
- Can you fix this?

Recurrence equation

Let $O(i,l)$ with $i \in [1..n]$, $l \in [2..k]$ denote the optimal value of the problem on the prefix sequence $[s_1, ..., s_i]$ using at most l partitions. $O(i,j) =$

$$1: \max\left(O(i-1,l), O(i-1,l-1) + s_i\right)$$

$$2: \min\left(O(i-1,l), O(i-1,l-1) + s_i\right)$$

$$3: \max_{j \in [1..i-1]} \min\left((s_{j+1} + s_{j+2} + \ldots + s_i), O(j,l-1)\right)$$

$$4: \min_{j \in [1..i-1]} \max\left((s_{j+1} + s_{j+2} + \ldots + s_i), O(j,l-1)\right)$$

$$5: \min_{j \in [1..i-1]} \max\left((s_{i+1} + s_{i+2} + \ldots + s_j), O(j,l-1)\right)$$

Explanation

$[\boxed{s_1, \ldots, s_j}, \boxed{s_{j+1}, \ldots, s_i}, \ldots, s_n]$

O(j,l-1) $\sum$

max

Max is necessary in case the last partition $[s_{j+1}, \ldots, s_i]$ is the heaviest one

$$O(i,l) = \min_{j \in [1..i-1]} \boxed{\max\left((s_{j+1} + s_{j+2} + \ldots + s_i), O(j, l-1)\right)}$$

Base case

- $O(i,l)$ $i \in [1..n]$ and $l \in [1..k]$.

- The base case is the one with one partition:
  - $O(i,1) = s1 + \ldots + si$

- And the one with the sequence of length 1
  - $O(1,l) = s1$ for all l in $[1..k]$.

- And this is not all, in your answer don't forget to characterize the optimal solution…
  - **The optimal solution is O(n,k)**

Table-Based Implementation

- O(5,2) = min(max(10,5),max(6,4+5),max(3,(3+4+5),max(1,(2+3+4+5))

- O(5,2) = min(10,9,12,14) = 9

k/s	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
1	1	3	6	10	15	21	28	36	45
2	1				**?**				
3	1								

$$O(i,l) = \min_{j\in[1..i-1]} \max\left((s_{j+1}+s_{j+2}+\ldots+s_i), O(j,l-1)\right)$$

Time-Complexity Analysis

$$O(i,l) = \min_{j \in [1..i-1]} \max\left((s_{j+1} + s_{j+2} + \ldots + s_i), O(j, l-1)\right)$$

- Time complexity to fill in the table ?

Final (Scala) Code

```scala
val S = Array(1, 2, 6, 3, 1, 4, 5, 6, 7, 8, 5)

val k = 4
val n = S.size
val O = Array.ofDim[Int](k,n)
// Base cases
for (k <- 0 until k) O(0)(k) = S(0)
for (i <- 1 until n) O(0)(i) = O(0)(i-1) + S(i)

// O(n^2) pre-processing
val prefixSum = Array.tabulate(n){i => S.take(i+1).sum}

def sumInterval(start: Int, end: Int) =
          prefixSum(end)-prefixSum(start-1)

// Recurrence
for (l <- 1 until k) {
  for (i <- 1 until n) {
    O(l)(i) = (for (j <- 0 to i-1) yield
                 {sumInterval(j+1,i) max O(l-1)(j) }).min
  }
}
println("optimal solution:"+O(k-1)(n-1))
```

What about this solution?

```scala
val S = Array(1, 2, 6, 3, 1, 4, 5, 6, 7, 8, 5)


def O(i: Int, l: Int): Int = {
  if (i == 0 || l == 1) {
    sumInterval(0, i)
  } else {
   (0 to i-1).map(j => sumInterval(j+1, i) max O(j, l - 1)).min
  }

}
```

How do you grade it ? What is the time complexity?

What about this one?

```
val S = Array(1, 2, 6, 3, 1, 4, 5, 6, 7, 8, 5)
val cache = collection.mutable.Map.empty[(Int, Int), Int]


def O(i: Int, l: Int): Int = {
  if (i == 0 || l == 1) {
    sumInterval(0, i)
  } else {
   (0 to i-1).map(j => sumInterval(j+1, i) max O_(j, l - 1)).min
  }

}

def O_(i: Int, l: Int) = cache.getOrElseUpdate((i, l), O(i, l))

println("optimal solution:" + O(n - 1,k))

println(cache.mkString(« ,"))
```

How do you grade it ? What is the time complexity?

Case study 2 (previous bonus question)

Question 5: Dynamic Programming [3pt]

The longest increasing subsequence problem is to find a subsequence of a given sequence in which the subsequence's elements are in in increasing order and in which the subsequence (not necessarily contiguous) is as long as possible. Example: (0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15) a longest increasing subsequence is (0, 2, 6, 9, 13, 15).

1. Explain a dynamic programming algorithm (detail the recurrence equations/expressions) to solve this problem and justify why it is correct?

2. Give and justify the time complexity of your algorithm ? Run and depict your algorithm on the given example.

Case Study 3: Longest common subsequence

- This algorithm is implemented in the « diff » unix utility
- LCS(A**GGTAB**,**GXTXAYB**) = 4 (GTAB)
- Solution is not always unique:
  - LCS(ABC,ACB) = 2 either for AB or AC
- LCS(AG**TAB**,**GXTXAYB**) = 4

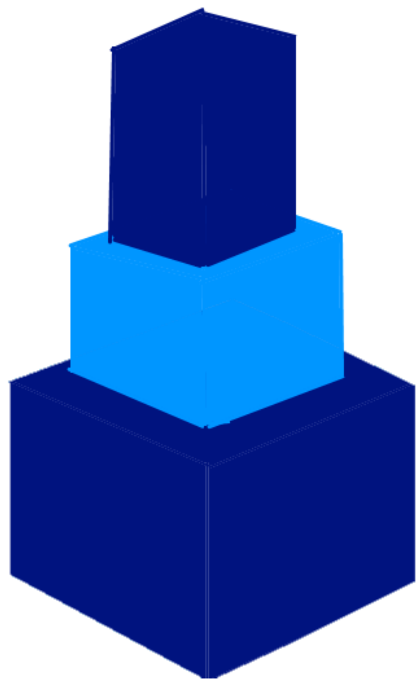
DB: LCS

- Let the input sequences be X[0..m-1] and Y[0..n-1]
- Define a dynamic program to compute the length of the longest common subsequence. Write the recurrence equations and the source code to solve the dynamic program.
- Hint: define LCS(X[0..i],Y[0..j]).
- What is the time-complexity?
- What if we have to find the longest common subsequence of 3 input sequences?

Case Study 4 (January 2016)

You are given a set of n types of rectangular 3-D boxes, where the ith box has height h(i), width w(i) and depth d(i) (all real numbers).

You want to create a stack of boxes which is as tall as possible, but you can only stack a box on top of another box if the dimensions of the 2-D base of the lower box are each strictly larger than those of the 2-D base of the higher box.



Box Stacking

Of course, you can rotate a box so that any side functions as its base. It is also allowable to use multiple instances of the same type of box.

Question:

What is the highest height given those four boxes?

(1,3,4),(4,2,1),(6,4,2),(1,7,1)

DP: Box-Stacking

- Hint for the recurrence equations, the dynamic program, the code, etc.

- What is the time complexity?

Box-Stacking problem

- Can you cast this problem to a graph problem that is well solved by dynamic programming (longest path in a DAG).

Bonus Challenge: Matrix Multiplication (+0.5)

- Suppose you want to multiply the matrices A x B x C x D of dimension 40x20, 20x300, 300x10, and 10x100
- Multiplying a m x n with a n x p takes m.n.p multiplications.
- (((AxB)xC)xD) takes how many operations?
- (Ax(BxC)xD) takes how many operations?
- How can you use dynamic programming to reduce as much as possible the number of operations?
- Prepare a small presentation for next lecture (send me the slides before) + a demo (with source-code).