CS 218 : Design and Analysis of Algorithms

Lecture 12: Array Partitioning Problems

Lecturer: *Sundar Vishwanathan*
Computer Science & Engineering Indian Institute of Technology, Bombay

1 Writing Recursive Procedures

This section is only for those students having trouble with the first step: writing recursive procedures. Others may skip this.

Often, you freeze on seeing a problem. At such times you should find things to do to get things moving. One is to try small examples. The other is to begin book-keeping. Begin by giving your procedure a name and clearly write down the parameters it will take as input. Indicate what your procedure will return. The procedure must ideally be recursive: in that when you recurse you should be calling the procedure with smaller inputs. In other words, the size of the input, as defined by the parameters, must go down. Make sure that you cover all possibilities during the recursion. The procedure is brute-force in that sense. Correctness of your procedure should be obvious.

Understand the input parameters, and infer what the input will be in subsequent stages of recursion. This will help you decide what the distinct recursive calls are. Often unrolling it twice is enough to understand this.

Some find it easier to write recursive programs by first having a mental picture of (and thinking about) what an optimum solution looks like. The other way you can work with is to imagine that somebody can give you solutions to all smaller inputs. How do you use this information to construct a solution to the current input? The queries you ask the other person forms the recursive calls.

2 Array Partitioning Problems

The simplest such problems is the following. You have given array A of integers and a bound B. Write a procedure to decide if you can partition it into at most k contiguous parts such that each part has sum at most B.

We work with the following generalization since the solution is roughly the same. The input is an $n \times n$ penalty matrix P and a positive constant c. The entry $P(i, j)$ stores the penalty that you incur if $[i, j]$ is one of the intervals in the partition. The problem is to divide the array into intervals of consecutive elements such that total penalty for the partition is minimum. The total penalty of a partition is the sum of the penalties associated with each interval in the partition plus c times the number of partitions minus one. Intuitively it costs c each time you create a new piece.

We call our procedure MinPartition$[1, \ldots, n]$. How do we recurse? There are exactly n choices for what the first partition can be. This then will yield the recursion. We will recurse over each of these n choices and choose the minimum of them. The optimum has to be one of these. Notice how easily correctness is guaranteed.

For each $i < n$, the recursive call will be $P(1, i) + c +$ MinPartition$[i + 1, \ldots, n]$. For $i = n$ it will be $P(1, i)$. *Exercise:* Prove correctness.

What are the distinct procedure calls? It pays to unroll the recursion once or twice more if in doubt. Unrolling the recursion twice you notice that all calls are with arguments of the form $[j, \ldots, n]$, for some j. So, the there is one distinct call per suffix of the array. We write the recurrence for a generic step which has parameters as the interval $[i, i+1, \ldots, n]$.

The recursion for the generic procedure is:

$$\min\{P[i,n], \min_{j<n} P[i,\ldots,j] + c + M\text{inPartition}[j+1,\ldots,n].\}$$

Exercise. Write the base cases.

We declare a table $T[i]$ with one entry for each distinct procedure call. What will $Y(i)$ store finally?

Exercise. Write the complete procedure which initialises the table and then, during the recursion, writes into each table entry exactly once. Analyse your procedure.