Introduction to Algorithms: 6.006
Massachusetts Institute of Technology — November 16th, 2010
Professors Konstantinos Daskalakis and Patrick Jaillet — Handout 6

Problem Set 6

This problem set is divided into two parts: Part A problems are theory questions, and Part B problems are programming tasks.

Part A questions are due **Tuesday, November 30th** at **11:59PM**.

Part B questions are due **Thursday, December 2nd** at **11:59PM**.

Solutions should be turned in through the course website. Your solution to Part A should be in PDF form using LaTeX or scanned handwritten solutions. Your solution to Part B should be a valid Python file, together with one PDF file containing your solutions to the two theoretical questions in Part B.

Templates for writing up solutions in LaTeX are available on the course website.

Remember, your goal is to communicate. Full credit will be given only to the correct solution which is described clearly. Convoluted and obtuse descriptions might receive low marks, even when they are correct. Also, aim for concise solutions, as it will save you time spent on write-ups, and also help you conceptualize the key idea of the problem.

Part A: Due Tuesday, November 30th

1. **(25 points)** Placing Parentheses

   You are given an arithmetic expression containing n real numbers and $n-1$ operators, each either $+$ or $\times$. Your goal is to perform the operations in an order that maximizes the value of the expression. That is, insert $n-1$ pairs of parentheses into the expression so that its value is maximized.

   For example:

   - For the expression $6 \times 3 + 2 \times 5$, the optimal ordering is to add the middle numbers first, then perform the multiplications: $((6 \times (3 + 2)) \times 5) = 150$.
   - For the expression $0.1 \times 0.1 + 0.1$, the optimal ordering is to perform the multiplication first, then the addition: $((0.1 \times 0.1) + 0.1) = 0.11$.
   - For the expression $(-3) \times 3 + 3$, the optimal ordering is $((-3) \times 3) + 3) = -6$.

   (a) **(10 points)** Clearly state the set of subproblems that you will use to solve this problem.

   Solution:

   First, we define some notation. Denote the numbers by $a_1, a_2, \ldots, a_n$, and the operators by $op_1, op_2, \ldots, op_{n-1}$. Let $E_{i,j}$ for $1 \leq i \leq j \leq n$ be the subexpression $a_i\ op_i\ \ldots\ op_{j-1}\ a_j$ of the input expression $E_{1,n} = a_1\ op_1\ \ldots\ op_{n-1}\ a_n$.

   We will have two types of subproblems:

- let $M[i, j]$, for $1 \leq i \leq j \leq n$ be the *maximum* value obtainable from $E_{i,j}$ (by inserting parentheses);
- similarly, let $m[i, j]$, for $1 \leq i \leq j \leq n$ be the *minimum* value obtainable from $E_{i,j}$.

Comment: we must keep track of both the minimum and the maximum because the maximal value of an expression may result from multiplying two negative subexpressions.

(b) **(10 points)** Write a recurrence relating the solution of a general subproblem to solutions of smaller subproblems.

Solution:

To compute the value $M[i, j]$ for some i and j, we note that if $i = j$ then $M[i, i] = E_{i,i} = a_i$. Otherwise, i.e. if $i < j$, we consider the *last*, say k^*-th, operator that was applied in an evaluation of $E_{i,j}$ that yields the maximal value.

This operator splits $E_{i,j}$ into two subexpressions E_{i,k^*} and $E_{k^*+1,j}$, i.e. we have

$$E_{i,j} = E_{i,k^*} \mathit{op}_{k^*} E_{k^*+1,j}$$

and both E_{i,k^*} and $E_{k^*+1,j}$ must have been evaluated before applying op_{k^*}. However, since op_{k^*} is either an addition or multiplication operator, in this optimal evaluation E_{i,k^*} must have been evaluated to either maximal ($M[i, k^*]$) or minimal ($m[i, k^*]$) possible value, and similarly $E_{k^*+1,j}$ must have yielded either $M[k^* + 1, j]$ or $m[k^* + 1, j]$. This implies that

$$k^* = \text{argmax}_{i \leq k \leq j-1} Eval_{\max}[i, j, k],$$

where

$$\begin{aligned} Eval_{\max}[i, j, k] = \max\{ & M[i, k] \; \mathit{op}_k \; M[k + 1, j], \\ & M[i, k] \; \mathit{op}_k \; m[k + 1, j], \\ & m[i, k] \; \mathit{op}_k \; M[k + 1, j], \\ & m[i, k] \; \mathit{op}_k \; m[k + 1, j]\}. \end{aligned}$$

Thus the recurrence relation for computation of $M[i, j]$ is given by:

$$M[i, j] = \begin{cases} a_i & \text{if } i = j \\ \max_{i \leq k \leq j-1} Eval_{\max}[i, j, k] & \text{otherwise} \end{cases}.$$

Analogous reasoning gives us a recurrence relation for computation of $m[i, j]$:

$$m[i, j] = \begin{cases} a_i & \text{if } i = j \\ \min_{i \leq k \leq j-1} Eval_{\min}[i, j, k] & \text{otherwise} \end{cases},$$

where

$$\begin{aligned} Eval_{\min}[i,j,k] = \min\{&M[i,k]\ op_k\ M[k+1,j], \\ &M[i,k]\ op_k\ m[k+1,j], \\ &m[i,k]\ op_k\ M[k+1,j], \\ &m[i,k]\ op_k\ m[k+1,j])\}. \end{aligned}$$

(c) **(5 points)** Analyze the running time of your algorithm, including the number of subproblems and the time spent per subproblem.

Solution:

First, we note that in our recurrence relations the subproblems to which we are reducing evaluation of $M[i,j]$ and $m[i,j]$ correspond to expressions of strictly smaller length. Thus we know that our recursion will eventually terminate by reaching base cases.

Comment: Even though establishing such termination property is usually quite simple, it constitutes an important step of application of dynamic programming technique and thus should not be completely ignored. (Usually, one-sentence explanation suffices to establish it.)

Now, we note that $M[1,n]$ is the answer that we want to compute. To bound the time needed to compute $M[1,n]$ using the above recurrence relations and memoization technique, we note that the total number of subproblems we are dealing with is $O(n^2)$. This is so, since there is at most $O(n^2)$ pairs (i,j) such that $1 \leq i \leq j \leq n$, and we have only two types of subproblem that are indexed by such pairs. Furthermore, computation of $M[i,j]$ or $m[i,j]$ takes $O(i-j) = O(n)$ time (assuming the needed subproblems are already computed). Therefore, since we need to compute each of $O(n^2)$ subproblems, the resulting total running time of the algorithm is $O(n^3)$.

Grading:

The base grading scale was as follows:

- (25/25) Correct solution;
- (18/25) Solution similar to the above, but not keeping track of the minimums, i.e. one that worked correctly if all the numbers were non-negative;
- (10/25) Any solution that used a reasonable (albeit incorrect) recurrence relation based on subproblems $M[i,j]$.

Sometimes there were penalties applied for erroneous or missing explanations of some key steps. E.g. a penalty of up to -5 points was applied for missing or incorrect running time analysis.

2. **(25 points)** Pots of Gold

There are N pots of gold arranged linearly. Alice and Bob are playing the following game. They take alternate turns, and in each turn they remove (and *win*) either of the two pots at the two ends of the line. Alice plays first. Given the amount of gold in each pot, design an algorithm to find the maximum amount of gold that Alice can assure herself of winning.

(a) **(10 points)** Clearly state the set of subproblems that you will use to solve this problem.

Solution: We will represent the pots as an array P with $P[i]$, for $1 \leq i \leq N$, equal to the amount of gold in i-th pot. Our subproblem will be $V[i, j]$, for $1 \leq i \leq j \leq N$, the maximum amount of gold that Alice can assure herself of winning when facing the (contiguous) subsequence $P[i : j]$ of the pots and being the first one to play.

(b) **(10 points)** Write a recurrence relating the solution of a general subproblem to solutions of smaller subproblems.

Solution: Consider a situation in which Alice is facing a sequence $P[i : j]$ of the pots, for $1 \leq i \leq j \leq N$, and it is her turn to move.

If $i = j$ then $V[i, i] = P[i]$, since she can just grab the only remaining pot of value $P[i]$. If $i + 1 = j$, she knows that whichever among the two pots of values $P[i]$ and $P[i + 1]$ she grabs the other one will be removed by Bob. So, $V[i, i+1] = \max\{P[i], P[i+1]\}$.

Now, consider the case of $i + 1 < j$. Alice has an option of removing either the i-th or the j-th pot.

If she decides to remove i-th pot then she gains $P[i]$ gold and Bob has to make his move while facing the sequence $P[i + 1 : j]$. During his move he can remove either $(i + 1)$-th or j-th pot, which would leave Alice facing the sequence $P[i + 2 : j]$ and $P[i + 1 : j - 1]$ respectively. Note that at this point, it would be Alice's turn again so the maximum amount of gold she could assure herself of winning would be $V[i+2 : j]$ and $V[i+1 : j-1]$ respectively. So, summarizing, if Alice decides to remove i-th pot, she can be sure of winning either $P[i]+V[i+2 : j]$ or $P[i]+V[i+1 : j-1]$ *depending on Bob's choice*. Since we don't know anything about Bob's strategy, the only amount of gold Alice can be sure of winning after removing i-th pot is the *minimum* of these two values.

Analogous reasoning implies that if Alice decides to remove j-th pot then the maximum amount of gold she can be sure of winning is $\min\{P[j] + V[i+1 : j-1], P[j] + V[i : j-2]\}$. So, since Alice can choose between the removal of the i-th and the removal of the j-th pot, her guaranteed amount $V[i, j]$ of gold to be won in this situation is

$$V[i, j] = \max\{P_1[i, j], P_2[i, j]\},$$

where

$$P_1[i, j] = \min\{P[i] + V[i + 2 : j], P[i] + V[i + 1 : j - 1]\},$$

and

$$P_2[i, j] = \min\{P[j] + V[i + 1 : j - 1], P[j] + V[i : j - 2]\}.$$

This leads to the following recurrence relation for $V[i, j]$:

$$V[i,j] = \begin{cases} P[i] & \text{if } i = j \\ \max\{P[i], P[i+1]\} & \text{if } i+1 = j \\ \max\{P_1[i,j], P_2[i,j]\} & \text{otherwise} \end{cases} .$$

Comment: Note that the above reasoning is a bit subtle. In particular, we are not allowed to assume anything about Bob's behavior, i.e. he does not need to follow a strategy that maximizes his gain. Of course, one might reduce the reasoning to the cases when he actually follows such a strategy[1]*, but such argument must be made explicitly.*

Note that once such an argument is made, one can develop yet another (more concise) recurrence relation for $V[i,j]$:

$$V[i,j] = \begin{cases} P[i] & \textit{if } i = j \\ SumP[i:j] - \min\{V[i+1:j], V[i:j-1]\} & \textit{otherwise} \end{cases},$$

where the $SumP[i:j]$ is the total amount of gold in subsequence $P[i:j]$, and this is computed efficiently using memoization.

(c) **(5 points)** Analyze the running time of your algorithm, including the number of subproblems and the time spent per subproblem. Hint: your overall algorithm should be $O(N^2)$.

Solution:

First, we note that in our recurrence relations the subproblems to which we are reducing evaluation of $V[i,j]$ correspond to strictly smaller number of pots. Thus we know that our recursion will eventually terminate by reaching base cases.

Comment: See comment to problem 1 (c).

Note that $V[1,N]$ is the final answer we are looking for. We proceed to bounding the running time of our memoization-based evaluation of the recurrence relation presented above. There is $O(N^2)$ subproblems $V[i,j]$ since there is at most that many pairs of indices (i,j) such that $1 \le i \le j \le N$. Computation of a single subproblem (assuming all the required subproblems are already computed) can be performed in $O(1)$ time. Therefore, the total running time is $O(N^2)$, as desired.

Grading:

The base grading scale was as follows:

- (25/25) Correct solution;

[1]Such reduction might, for instance, go along the lines of utilizing the fact that we are dealing with a zero-sum game (i.e. Alice's loss is Bob's gain and vice versa) and the maximum guaranteed gain of Alice must match the total amount of gold in the game minus the maximum guaranteed gain of Bob.

- (18/25) Solution that is based on a recurrence relation that considers all the four possible sequence of moves of Alice and Bob, but takes the maximum of the corresponding outcomes instead of maxmin as above – this corresponds to a situation when Bob is cooperating with Alice;
- (10/25) Solution that was based on recurrence relation that looked only one step ahead and/or was applying greedy strategy.

Sometimes there were penalties applied for erroneous or missing explanations of some key steps. E.g. a penalty was applied if the reasoning behind the recurrence relation (which is the crux of this problem) was not satisfactory, or any unjustified assumptions about Bob's behavior were made.