CS 124 Homework 4: Spring 2022

Your name: Lev Kruglyak

Collaborators: Swati Goel, Sahil Kuchlous

No. of late days used on previous psets: 6
No. of late days used after including this pset: 8

Homework is due Wednesday at midnight ET. You are allowed up to **twelve** (college)/**forty** (extension school) late days through the semester, but the number of late days you take on each assignme nt must be a nonnegative integer at most **two** (college)/**four** (extension school).

Try to make your answers as clear and concise as possible; style may count in your grades. Assignments must be submitted in pdf format on Gradescope. If you do assignments by hand, you will need to scan in your results to turn them in.

You can collaborate with other students that are currently enrolled in this course in brainstorming and thinking through approaches to solutions but you should write the solutions on your own: you must wait one hour after any collaboration or use of notes from collaboration before any writing in your own solutions that you will submit.

For all homework problems where you are asked to give an algorithm, you must prove the correctness of your algorithm and establish the best upper bound that you can give for the running time. Generally better running times will get better credit; generally exponential time algorithms (unless specifically asked for) will receive no or little credit. You should always write a clear informal description of your algorithm in English. You may also write pseudocode if you feel your informal explanation requires more precision and detail, but keep in mind pseudocode does NOT substitute for an explanation. Answers that consist solely of pseudocode will receive little or not credit. Again, try to make your answers clear and concise.

Problem 1. Consider an algorithm for integer multiplication of two n-digit numbers where each number is split into three parts, each with $n/3$ digits. (You may assume that n is a power of 3.)

(a) **(15 points)** Design and explain such an algorithm, similar to the integer multiplication algorithm presented in class. Your algorithm should describe how to multiply the two integers using only six multiplications on the smaller parts (instead of the straightforward nine).

(b) **(7 points)** Determine the asymptotic running time of your algorithm. Would you rather split it into two parts or three parts?

(c) **(7 points)** Suppose you could use only five multiplications instead of six. Then determine the asymptotic running time of such an algorithm. In this case, would you rather split it into two parts or three parts?

(d) **(0 points, optional)** Find a way to use only five multiplications on the smaller parts.

(e) **(0 points, optional)** Generalize the previous subproblem to when the two initial n-digit numbers are split into k parts, each with n/k digits. (You may assume that n is a power of k.)

Hint: also consider multiplication by a constant, such as 2; note that multiplying by 2 does not count as one of the five multiplications. You may need to use some linear algebra.

(a) (For this problem, we don't consider the sign bit to be a digit, so -3 is a one digit integer in base 10) Suppose we are asked to multiply two n digit numbers a and b, where n is a power of three. We can write $a = a_0 + a_1 B^{n/3} + a_2 B^{2n/3}$ and $b = b_0 + b_1 B^{n/3} + b_2 B^{2n/3}$ where B is the base and a_i, b_i are all $n/3$ digit numbers. Then

$$\begin{aligned} ab &= (a_0 + a_1 B^{n/3} + a_2 B^{2n/3})(b_0 + b_1 B^{n/3} + b_2 B^{2n/3}) \\ &= c_0 + c_1 B^{n/3} + c_2 B^{2n/3} + c_3 B^{n} + c_4 B^{4n/3} \end{aligned}$$

where $c_0 = a_0 b_0$, $c_1 = a_0 b_1 + a_1 b_0$, $c_2 = a_0 b_2 + a_1 b_1 + a_2 b_1$, $c_3 = a_2 b_1 + a_1 b_2$, and $c_4 = a_2 b_2$. We claim that we only need to calculate the products $P_0 = a_0 b_0, P_1 = a_1 b_1, P_2 = a_2 b_2$, and $Q_0 = (a_0 - a_1)(b_1 - a_0), Q_1 = (a_1 - a_2)(b_2 - a_1), Q_2 = (a_2 - a_0)(b_0 - b_2)$. These are all products of two numbers with $n/3$ digits. (again, since we don't count the sign bit, $a_i - a_j$ always has $n/3$ digits) The c_i can then be expressed:

$$\begin{aligned} c_0 &= P_0 \\ c_1 &= Q_0 + P_0 + P_1 \\ c_2 &= Q_2 + P_0 + P_1 + P_2 \\ c_3 &= Q_1 + P_1 + P_2 \\ c_4 &= P_2 \end{aligned}$$

(b) Let $T(n)$ be the time to multiply two n digit numbers using our algorithm. Since we only need to perform 6 multiplications of $n/3$ digit numbers and some constant number of additions, we can establish a recurrence for $T(n)$:

$$T(n) = 6T(n/3) + O(1).$$

Then by the master recurrence theorem we get $T(n) \in \Theta(n^{\log_3 6}) = \Theta(n^{1.63})$. It would not be more efficient to switch to splitting numbers into 3 parts, because splitting numbers into 2 parts gives $T(n) \in \Theta(n^{\log_2 3}) = \Theta(n^{1.58})$ which is faster.

(c) Here we can set up a similar recurrence:

$$T(n) = 5T(n/3) + O(1)$$

and by the master theorem we get $T(n) \in \Theta(n^{\log_3 5}) = \Theta(n^{1.46})$ which is faster than the two splitting algorithm, so it would be better to split into 3 and use 5 multiplications.

Problem 2.

(a) **(25 points)** Suppose we want to print a paragraph neatly on a page. The paragraph consists of words of length $\ell_1, \ell_2, \ldots, \ell_n$. The maximum line length is M. (Assume $\ell_i \leq M$ always.) We define a measure of neatness as follows. The extra space on a line (using one space between words) containing words ℓ_i through ℓ_j is $M - j + i - \sum_{k=i}^{j} \ell_k$. The penalty is the sum over all lines **except the last** of the **cube** of the extra space at the end of the line. This has been proven to be an effective heuristic for neatness in practice. Find a dynamic programming algorithm to determine the neatest way to print a paragraph. Of course you should provide a recursive definition of the value of the optimal solution that motivates your algorithm.

(b) **(20 points)** Determine the minimal penalty, and corresponding optimal division of words into lines, for the text of this question, from the first 'Determine' through the last 'correctly.)', for the cases where M is 40 and M is 72. Words are divided only by spaces (and, in the pdf version, linebreaks) and each character that isn't a space (or a linebreak) counts toward the length of a word, so the last word of the question has length eleven, counting the period and the right parenthesis. (You can find the answer by whatever method you like, but we recommend coding your algorithm from the previous part. You don't need to submit code.) (The text of the review may be easier to copy-paste from the tex source than from the pdf. If you copy-paste from the pdf, check that all the characters show up correctly.)

(a) Define the *penalty* of a sequence of words $w_i, \ldots, w_j$ as:

$$p(i,j) = \begin{cases} \left(M - j + i - \sum_{k=i}^{j} \ell_k\right)^3 & \text{if this quantity is nonnegative} \\ \infty & \text{otherwise} \end{cases}$$

Given some sequence of lines, represented by a sequence of indices $1 \leq I_1 \leq I_2 \leq \cdots \leq I_m \leq n$ where the line breaks are inserted after the word w_{I_i}, we define the *total penalty* of this line break decomposition to be

$$P(I_1, \ldots, I_m) = p(1, I_1) + \sum_{i=1}^{m-1} p(I_i, I_{i+1}).$$

Notice that we do not include the penalty of the last line $w_{I_m}, \ldots, w_n$ in the total penalty computation. This is exactly the penalty function defined in the problem statement, our goal is to find some line break sequence which minimizes this penalty function. For any $1 \leq i \leq n$, let $D[i]$ be the minimal penalty which can be achieved by adding line breaks to the sequence of words $w_i, \ldots, w_n$. As with typical dynamic programming problems, $D[i]$ satisfies the following recurrence:

$$D[i] = \min_{i<j\leq n} \left(p(i,j) + D[j+1]\right)$$

where $D[n+1] = 0$. If we want to actually determine where to place the line breaks, we can keep track of which j contributed the minimum to $D[i]$ in some separate array $A[i]$. The minimum penalty for the whole sequence of words is simply $D[1] = P(A[1], A[A[1]+1], \ldots)$, so the minimal lines are $\{w_1, \ldots, w_{A[01]}\}$, $\{w_{A[1]+1}, \ldots, w_{A[A[1]+1]}\}$, etc. Now, to see why this recurrence works, note that j just keeps track of where the first line ends. For each of the possible places to end the line, the optimal penalty for the rest of the text won't be affected by the addition or removal of this first line, if adding it causes there to be a more optimal splitting, then we could've made this split before adding the line. So if we loop over all possible line endings for the first line and choose the smallest penalty value, this will in fact be optimal.

To efficiently calculate $p(i,j)$ we can start by precalculating a word length list, $w[i]$, defined as $w[i] = \sum_{k=0}^{i} \ell_k$. This can be done in $O(n)$ recursively using the rule $w[i] = \ell_i + w[i-1]$ using only $O(n)$ space. Then we have a simple expression for $p(i,j)$:

$$p(i,j) = (M - j + i - w[j] - w[i-1])^3.$$

So after the $O(n)$ preprocessing step we can compute $p(i,j)$ in $O(1)$. Next, to compute $D[i]$, we can just start by setting $D[n+1]=0$, and then computing $D[n], D[n-1]$, and so on until we reach $D[1]$. This takes $O(n^2)$ because we must calculate a minimum over $O(n)$ elements every step, and we do this n times. So overall, our dynamic programming algorithm takes $O(n^2)$, and uses only $O(n)$ memory.

(b) For $M=40$, we get a penalty of 1116, achieved by the following line break decomposition:

```
Determine the minimal penalty, and
corresponding optimal division of
words into lines, for the text of this
question, from the first 'Determine'
through the last 'correctly.)', for
the cases where M is 40 and M is 72.
Words are divided only by spaces (and,
in the pdf version, linebreaks) and
each character that isn't a space (or a
linebreak) counts toward the length of
a word, so the last word of the question
has length eleven, counting the period
and the right parenthesis.  (You can find
the answer by whatever method you like,
but we recommend coding your algorithm
from the previous part.  You don't need
to submit code.)  (The text of the review
may be easier to copy-paste from the
tex source than from the pdf.  If you
copy-paste from the pdf, check that all
the characters show up correctly.)
```

For $M=72$, we get a penalty of 885, achieved by the following line break decomposition:

```
Determine the minimal penalty, and corresponding optimal division
of words into lines, for the text of this question, from the first
'Determine' through the last 'correctly.)', for the cases where M is 40
and M is 72.  Words are divided only by spaces (and, in the pdf version,
linebreaks) and each character that isn't a space (or a linebreak)
counts toward the length of a word, so the last word of the question
has length eleven, counting the period and the right parenthesis.  (You
can find the answer by whatever method you like, but we recommend coding
your algorithm from the previous part.  You don't need to submit code.)
(The text of the review may be easier to copy-paste from the tex source
than from the pdf.  If you copy-paste from the pdf, check that all the
characters show up correctly.)
```

Problem 3. **(25 points)** Consider the following string compression problem. We are given a dictionary of m strings, each of length at most k. We want to encode a data string of length n using as few dictionary strings as possible. For example, if the data string is *bababbaababa* and the dictionary is $(a, ba, abab, b)$, the best encoding is $(b, abab, ba, abab, a)$. Give an $O(nmk)$ algorithm to find the length of the best encoding or return an appropriate answer if no encoding exists.

Let D be the set of all words in the dictionary, and for every word $w \in D$, let $\ell(w)$ be its length. We'll use the notation $[i : j]$ to denote the substring of the target string ranging from i to j inclusive. Similarly, $[i :]$ is shorthand for $[i : n]$, etc. Lastly, let $s(w, [i : j])$ be defined:

$$s(w, [i : j]) = \begin{cases} 1 & w = [i : i + \ell(w) - 1] \\ \infty & \text{otherwise} \end{cases}$$

so $s(w, [i : j])$ returns a 1 only if the string $[i : j]$ starts with w. Now let $D[i]$ be the length of the best encoding for the string $[i :]$. Our goal is to calculate $D[0]$. To create a recurrence for $D[i]$, note that if we had some optimal encoding $\{w_1, w_2, \ldots, w_q\}$ for a string $[i :]$, then $\{w_2, \ldots, w_q\}$ must be an optimal encoding for the string $[i + \ell(w_1) - 1 :]$. This is because if we could find a better encoding for $[i + \ell(w_1) - 1 :]$, this would give us a better encoding for $[i :]$ as well after adding w_1 back to the front. This motivates the following recurrence:

$$D[i] = \min_{w \in D} \left(s(w, [i :]) + D[i + \ell(w)] \right).$$

So at a given index, we try using all possible dictionary words which can be inserted at the current position, and seeing which one gives the best encoding via the minimum function. Intuitively, we add words one by one to turn an optimal solution into a larger optimal solution. We can show by induction that at every step of the recurrence, we have an optimal solution. Note that since all dictionary words have length less than k, the time complexity of $s(w, [i :])$ is $O(k)$. Then the time complexity to calculate the minimum is $O(mk)$ since there are m elements in the dictionary. Finally, to calculate $D[0]$, we need to calculate $D[i]$ for all $i > 0$, so the total time complexity is $O(mnk)$ as desired. Lastly, note that if no encoding exists, $D[0]$ will be ∞.

Problem 4. **(25 points)** Given a positive integer n and, for each pair (i, j) with $0 \leq i \leq j < n$, an integer $X_{i,j} \in \{0, 1\}$, give an algorithm to find integers $C_{i,j} \in \{0, 1\}$ for each pair (i, j) with $0 \leq i \leq j < n$ such that:

(a) For all i, j, and k with $i \leq j \leq k$, if $C_{i,j} = 1$ and $C_{j,k} = 1$, then $C_{i,k} = 1$.

(b) For all i, j, and k with $i \leq j \leq k$, if $C_{i,k} = 1$, then $C_{i,j} = 1$ and $C_{j,k} = 1$.

(c) Subject to the above,

$$2 \sum_{X_{ij}=1} C_{ij} - \sum_{X_{ij}=0} C_{ij}$$

is as large as possible.

If we consider $X_{i,j}$ and $C_{i,j}$ as points with coordinates (i, j), we're restricted to an upper triangular region of the plane, since $i \leq j$. The conditions (a) and (b) imply that if $C_{i,j} = 1$, then the triangular region with vertex at i, j. So for any $0 \leq i \leq j < n$ and $d \in \{0, 1\}$, let $D[i, j, d]$ be the maximum cost (where the cost is summed over $i \leq k_1 \leq k_2 \leq j$) which can be achieved by setting $C_{i,j} = d$. This satisfies the recurrence

$$D[i,j,d] = \left(d \begin{cases} 2 & X_{i,j} = 1 \\ -1 & X_{i,j} = 0 \end{cases} \right) + \min \begin{cases} D[i,j-1,0] + D[i+1,j,0] \\ D[i,j-1,1] + D[i+1,j,0] \\ D[i,j-1,0] + D[i+1,j,1] \\ D[i,j-1,1] + D[i+1,j,1] - D[i+1,j-1,1] \end{cases} .$$

We can assume the array is zero padded to avoid out of bound issues. This recurrence works by checking all of the cases for the two points lying to the right and bottom of the root point; the value of the corner point has no effect on the maximum costs on the neighboring points. Notice we have to subtract $D[i + 1, j - 1, 1]$ in the last case to avoid double counting. Let's order the subproblems in increasing order by setting the size of $D[i, j, d]$ to $j - i$. Our base case then becomes solving $D[i, i, d]$, which is just equal to 0 if $d = 0$ and 2 if $X_{i,i} = 1$ and -1 if $X_{i,i} = 0$. Then solving $D[i, j, d]$ involves computing subproblems of sizes $j - i - 1$ and $j - i - 2$, so we can fully solve $D[i, j, d]$. Lastly, starting at $(i, j) = (0, n - 1)$, set $C_{i,j} = \text{argmin}_{d \in \{0,1\}} D[i, j, d]$, marking it as processed. If we set $C_{i,j} = 1$, then we set $C_{k_1,k_2} = 1$ for all $i \leq k_1 \leq k_2 \leq j$ and mark them processed. If we do this for all i, j, ordering by $j - i$ in descending order, we'll get a $C_{i,j}$ value that maximizes the cost function.