CS 621 - Assignment 4

Ali Hassani

November 10, 2022

1 Chapter 6, Exercise 6

In a word processor, the goal of "pretty-printing" is to take text with a ragged right margin, like this,

```
Call me Ishmael.
Some years ago,
never mind how long precisely,
having little or no money in my purse,
and nothing particular to interest me on shore,
I thought I would sail about a little
and see the watery part of the world.
```

and turn it into text whose right margin is as "even" as possible, like this.

```
Call me Ishmael. Some years ago, never
mind how long precisely, having little
or no money in my purse, and nothing
particular to interest me on shore, I
thought I would sail about a little
and see the watery part of the world.
```

To make this precise enough for us to start thinking about how to write a pretty-printer for text, we need to figure out what it means for the right margins to be "even." So suppose our text consists of a sequence of words, $W = w_1, w_2, ..., w_n$, where w_i consists of c_i characters. We have a maximum line length of L. We will assume we have a fixed-width font and ignore issues of punctuation or hyphenation.

A formatting of W consists of a partition of the words in W into lines. In the words assigned to a single line, there should be a space after each word except the last; and so if $w_j, w_{j+1}, ..., w_k$ are assigned to one line, then we should have

$$\left[\sum_{i=j}^{k-1}(c_i + 1)\right] + c_k \leq L.$$

We will call an assignment of words to a line *valid* if it satisfies this inequality. The difference between the left-hand side and the right-hand side will be called the *slack* of the line—that is, the number of spaces left at the right margin.

Give an efficient algorithm to find a partition of a set of words W into valid lines, so that the sum of the squares of the slacks of all lines (including the last line) is minimized.

Answer:

Subproblem:
$S(i, j)$ is the slack for a single line formed by $w_i, w_{i+1}, ..., w_j$ for nonnegative slacks, and ∞ for negative slacks (invalid assignment). It can be precomputed with $\mathcal{O}(n^3)$ and stored with $\mathcal{O}(n^2)$ space.
$R(i)$ is sum of squares of slacks of all lines formed by partitioning $w_1, w_2, ..., w_i$, subject to all lines being valid.

Recurrence:

$$R(i) = \begin{cases} 0 & \text{if } i \leq 0 \\ S(1,1)^2 & \text{if } i = 1 \\ \min_{j=1}^{i} \{R(j-1) + S(j,i)^2\} & \text{otherwise} \end{cases}$$

Desired output: $R(n)$.

2 Chapter 6, Exercise 19

You're consulting for a group of people (who would prefer not to be mentioned here by name) whose jobs consist of monitoring and analyzing electronic signals coming from ships in coastal Atlantic waters. They want a fast algorithm for a basic primitive that arises frequently: "untangling" a superposition of two known signals. Specifically, they're picturing a situation in which each of two ships is emitting a short sequence of 0s and 1s over and over, and they want to make sure that the signal they're hearing is simply an interleaving of these two emissions, with nothing extra added in.

This describes the whole problem; we can make it a little more explicit as follows. Given a string x consisting of 0s and 1s, we write x^k to denote k copies of x concatenated together. We say that a string x' is a repetition of x if it is a prefix of x^k for some number k. So $x' = 10110110110$ is a repetition of $x = 101$.

We say that a string s is an *interleaving* of x and y if its symbols can be partitioned into two (not necessarily contiguous) subsequences s' and s'', so that s' is a repetition of x and s'' is a repetition of y. (So each symbol in s must belong to exactly one of s' or s''.) For example, if $x = 101$ and $y = 00$, then $s = 100010101$ is an interleaving of x and y, since characters $1, 2, 5, 7, 8, 9$ form 101101—a repetition of x—and the remaining characters $3, 4, 6$ form 000—a repetition of y.

In terms of our application, x and y are the repeating sequences from the two ships, and s is the signal we're listening to: We want to make sure s "unravels" into simple repetitions of x and y.

Give an efficient algorithm that takes strings s, x, and y and decides if s is an interleaving of x and y.

Answer:

Given strings s, x, and y, where s is of length n, we form x' and y' to be repetitions of x and y respectively, both of exactly length n. The definition of *repetition* guarantees that only one such x' and y' exists. For any string z, we define $z[i]$ to be the i-th symbol in z and $z[i:j]$ to be the i-th through j-th symbols (including j-th).

Subproblem: We define $R(i,j)$ to be `True` IFF $s[1:i+j]$ is an interleaving of $x'[1:i]$ and $y'[1:j]$.

Recurrence:

$$R(i,j) = \begin{cases} 0 & \text{if } i \leq 0 \text{ OR } j \leq 0 \\ s[1] == x'[1] & \text{if } i = 1 \text{ AND } j = 0 \\ s[1] == y'[1] & \text{if } i = 0 \text{ AND } j = 1 \\ \\ \big(s[i+j] == x'[i] \;\&\&\; R(i-1,j)\big) & \\ \;\;|| \;\; \big(s[i+j] == y'[j] \;\&\&\; R(i,j-1)\big) & \text{otherwise} \end{cases}$$

Desired output: If any i and j covers all of s and results in a `True` value, then s is an interleaving of x and y.

$\max_{i=1}^{n} \left(R(i, n-i)\right)$.

3 (DPV) Chapter 6, Exercise 26

Consider the following version of the sequence alignment problem.
We have strings $X, Y \in \Sigma^*$, and the closeness of the alignment between $X = X_1X_2...X_n$ and $Y = Y_1Y_2...Y_m$ is determined by a scoring matrix δ of size $(|\Sigma|+1)\times(|\Sigma|+1)$, where the extra rows and columns are to accommodate gaps. We define a subproblem $AS(i, j)$ ("alignment score") as the score of the highest-scoring alignment of $X = X_1X_2...X_i$ and $Y = Y_1Y_2...Y_j$.

a. Give a recurrence for $AS(i, j)$.

b. What values assigned to δ describe the Longest Common Subsequence problem?

c. What values assigned to δ describe the Edit Distance problem?

Answer:

(a) recurrence:

$$AS(i,j) = \begin{cases} 0 & \text{if } i \leq 0 \text{ OR } j \leq 0 \\ \\ \max\big(R(i-1,j), R(i,j-1)\big) + \delta[X_i, Y_i] & \text{otherwise} \end{cases}$$

(b) Longest Common Subsequence: Delta should be set to the identity matrix, so that matched columns get a single point and unmatched ones get zero. In other words, for every $(a, b) \in \Sigma \times \Sigma$:

$$\delta[a,b] = \begin{cases} 1 & \text{if } a = b \\ 0 & \text{if } a \neq b \end{cases}$$

(c) Edit Distance: Assuming replace and remove have the same penalty: For every $(a, b) \in \Sigma \times \Sigma$:

$$\delta[a,b] = \begin{cases} 0 & \text{if } a = b \\ -1 & \text{if } a \neq b \end{cases}$$

4 (DPV) Chapter 6, Exercise 6

We define a multiplication operation table on three symbols a, b, c according to something like the table below, so that $ab = b$, $ca = a$, and so on. The goal is to determine whether a string of symbols, $s_1 s_2 ... s_n$ ($s_i \in \{a, b, c\}$) can be parenthesized in such a way so that their multiplication together equals a. Note that the operation is neither associative nor commutative (that is, it is possible that $ab \neq ba$ and $(ab)c \neq a(bc)$).

Table 1: multiplication table for a, b, c

	a	b	c
a	b	b	a
b	c	b	a
c	a	c	c

For example, if the input string is *bbbbac* then the answer should be **true** since $((b(bb))(ba))c = a$. *Hint*: define a subproblem $P[i, j, t]$, where $1 \leq i \leq j \leq n$ and $t \in \{a, b, c\}$. $P[i, j, t]$ will be **true** if it is possible to parenthesize $s_i s_{i+1} ... s_j$ in such a way that the multiplication in that order equals the symbol t, and **false** otherwise.

Answer:

Subproblem: We define $P(i, j, t)$ to be `True` only if it is possible to parenthesize $s_i s_{i+1} ... s_j$ in a way that it equals t.

Recurrence:

$$P(i,j,t) = \begin{cases} False & \text{if } i > j \text{ OR } i \leq 0 \\ s_i == t & \text{if } i = j \\ s_i s_{i+1} == t & \text{if } i = j - 1 \\ \\ \max\limits_{i \leq k < j} \max\limits_{\substack{p,q \in \{a,b,c\} \\ pq == t}} \big(P(i,k,p) P(k+1,j,q)\big) & \text{otherwise} \end{cases}$$

Desired output: $P(1, n, a)$.

5 Bones's Battery

Bones is investigating what electric shuttle is appropriate for his mom's school district vehicle. Each school has a charging station. It is important that a trip from one school to any other be completed with no more than K rechargings. The car is initially at zero battery and must always be recharged at the start of each trip; this counts as one of the K rechargings. There is at most one road between each pair of schools, and there is at least one path of roads connecting each pair of schools. Given the layout of these roads and K, compute the necessary range required of the electric shuttle.

Input
Input begins with a line with one integer T ($1 \leq T \leq 50$) denoting the number of test cases. Each test case begins with a line containing three integers N, K, and M ($2 \leq N \leq 100$, $1 \leq K \leq 100$), where N denotes the number of schools, K denotes the maximum number of rechargings permitted per trip, and M denotes the number of roads. Next follow M lines each with three integers u_i, v_i, and d_i ($0 \leq u_i, v_i < N$, $u_i \neq v_i$, $1 \leq d_i \leq 10^9$) indicating that road i connects schools u_i and v_i (0-indexed) bidirectionally with distance d_i.

Answer:

Subproblem: Define $D^k(i, j)$ to be the minimum distance between schools i and j, with a maximum of k intermediate schools.

Recurrence:

$$D_k(i,j) = \begin{cases} \infty & \text{if no road between } (i,j) \text{ AND } k = 0 \\ dist_{i,j} & \text{if road between } (i,j) \text{ AND } k = 0 \\ \min\left(D_{k-1}(i,j), D_{k-1}(i,k) + D_{k-1}(k,j)\right) & \text{otherwise} \end{cases}$$

Desired output: $\max_{i,j} D^K(i,j)$.